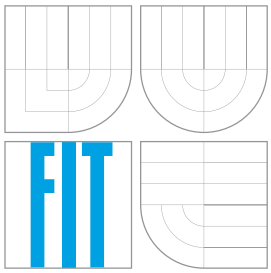
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

IMPLEMENTACE POKROČILÝCH BEZPEČNOSTNÍCH ALGORITMŮ DO DOMÁCÍCH ROUTERŮ

IMPLEMENTATION OF THE ADVANCED SECURITY ALGORITHMS INTO HOME ROUTERS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN ŠPINLER

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN KAŠTIL

BRNO 2011

Abstrakt

Práce se zabývá síťovými prvky, které slouží ke směrování paketů v malých sítích. Je zaměřena na dvě konkrétní zařízení, které obsahují procesor ARM. V práci je detailně popsán postup instalace operačního systému OpenWRT do těchto zařízení. Dále je zde proveden test výkonnosti těchto a dalších vybraných zařízení při směrování a test výkonnosti algoritmů používaných v síťové bezpečnosti. Součástí práce je též modul do operačního systému OpenWRT pro vyhledávání vzorů v síťovém provozu.

Abstract

This work deals with network devices designed for packet routing in small networks. It's focused on two particular devices with ARM processor. There is detailed description of an instalation of OpenWRT operating system to this devices. Next, there are results of tests of routing performance of these and some other selected devices and tests of performance of alogorithms used in network security. A module for OpenWRT operating system for pattern-matching in network traffic is also part of the work.

Klíčová slova

směrování, router, ARM, OpenWRT, Linux, počítačová síť, vyhledání vzorů

Keywords

routing, router, ARM, OpenWRT, Linux, computer network, pattern match

Citace

Martin Špinler: Implementace pokročilých bezpečnostních algoritmů do domácích routerů, diplomová práce, Brno, FIT VUT v Brně, 2011

Implementace pokročilých bezpečnostních algoritmů do domácích routerů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Kaštila. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal

.....
Martin Špinler
25.května 2011

Poděkování

Chtěl bych poděkovat Ing. Janu Kaštilovi za trpělivost a obětavost při konzultacích. Také bych rád poděkoval škole za vstřícnost při nákupu zařízení, které mi umožnily uskutečnit vytvoření této práce.

Obsah

Kapitola 1

Úvod

Možnost komunikovat je bezesporu jedním ze základních prvků lidské společnosti. Díky komunikaci si totiž můžeme vyměňovat důležité informace či znalosti, na kterých může záviset třeba i náš život. Podobně tomu je i ve světě počítačů – komunikace počítačů mezi sebou je nepostradatelnou součástí, zvlášť v dnešní době, kdy používáme různé internetové služby, multimediální servery, síťová datová úložiště a další.

Druhým základním prvkem jsou informace, které přenášíme právě pomocí komunikace. Lze dokonce říci, že jedno bez druhého ztrácí svůj význam, komunikace bez informací nedává žádný smysl. A aby počítače byly schopny komunikovat mezi sebou a přenášet informace (v oblasti digitální techniky je nazýváme data), bylo nutné specifikovat přenosové protokoly, určit datová média a navrhnout stukturu propojení počítačů. Této trojici říkáme počítačová síť a je to tedy označení pro prostředky, které provádí přenos dat mezi počítači.

Díky stálému nárůstu využití internetové sítě se zvyšují nároky hlavně na přenosovou kapacitu a minimální časovou prodlevu. S tím souvisí především výkonnost aktivních prvků sítě (to jsou fyzická zařízení, např. počítač nebo směřovač). Jednotlivé počítače, které máme doma, dnes již mají dostatečnou výkonnost, aby uspokojily naše síťové požadavky. Ovšem platí to také o spojovacích uzlech – prvcích, které agregují datové požadavky jednotlivých počítačů?

Klasický domácí směrovač (dále jen router) je ve své podstatě vestavěné zařízení obsahující určitý procesor, paměť RAM, paměť Flash a několik Ethernetových portů, popřípadě i Wi-Fi modul. Na tomto zařízení je spuštěný nějaký operační systém, který přijímá data ze síťových rozhraní, určitým způsobem je zpracuje a odesílá dále do sítě. Způsob zpracování dat se liší podle konfigurace systému, především se jedná o směrování paketů do cílových sítí. Ale v systému mohou být konfigurovány další služby, jako DHCP, Firewall a NAT a protože každá služba do jisté míry zatěžuje procesor, naskýtá se otázka, jaké množství dat dokáže procesor zpracovat.

Přitom nelze opomenout složitost různých algoritmů použitých při zpracování dat. Je třeba algoritmy rozebrat a teoreticky i prakticky ověřit dobu výpočtu na nějakém univerzálním systému. A protože ve většině případů požadujeme, aby algoritmus proběhl na vstupních datech co nejrychleji, snažíme se využít dostupné prostředky k tomu, abchychom výpočet urychlili. To je možné například použitím jiného algoritmu k výpočtu, jeho optimalizací, či akcelerací určité jeho části za použití nějaké speciální hardwarové jednotky.

Jakmile máme hotový algoritmus ke zpracování dat, můžeme jej začlenit do celého systému. Měli bychom znát systémovou strukturu a jeho rozhraní, které použijeme pro propojení s algoritmem. Většinou je třeba získat ze systému data, upravit je do formy vhodné pro algoritmus a předat systému výsledky.

Nakonec je potřeba celý systém otestovat z hlediska funkčnosti a podrobit výkonostním testům. U routeru, který přeposílá data mezi různými sítěmi, je ideální stav takový, že při plném provozu všech linek stíhá plnit všechny nastavené funkce. Ovšem jak si ukážeme, tento stav zdaleka neodpovídá stavu současných routerů, které běžně uživatelé používají ve svých domácnostech.

V této práci se tedy budeme zabývat domácími routery s procesorem ARM. V první části si vysvětlíme důležité pojmy a principy používané v počítačových síťích. Projdeme si bežnou funkcionalitu domácích routerů a uvedeme si konkrétní zařízení, se kterými budeme pracovat. Také obecně prozkoumáme architekturu procesorů ARM a specifika procesorů v našich zařízeních. Následovat bude kapitola s teoretickým rozborem algoritmů, se kterými se zde setkáme.

V praktické částí práce se seznámíme s operačním systémem OpenWRT, který je určen především pro routery a ukážeme si, jak tento systém spustit na vybraných zařízení. Vyzkoušíme pro systém OpenWRT vytvořit vlastní modul, který posléze rozšíříme o kompletní systém pro detekci vzorů.

Dále budou provedeny výkonstní testy: nejprve otestujeme výkonnost procesorů při běžném provozu, tj. při směrování paketů. Provedeme otestování vybraných algoritmů a nakonec shrneme dosažené výsledky a zamyslíme se nad možností pokračování práce nad zařízeními.

Kapitola 2

Síťová zařízení

Základem každé počítačové sítě jsou síťová zařízení. V této kapitole se tedy seznámíme se síťovými vrstvami a zařízeními, které nad nimi pracují. Podíváme se, jaké algoritmy a funkce jsou často implementované v běžných routerech. Také si představíme zařízení, o kterých budeme v této práci mluvit.

2.1 Síťové vrstvy a příslušná zařízení

V klasických sítích probíhá obousměrná komunikace zasíláním dat v paketech mezi cílovými zařízeními. Pokud rozebereme jeden paket tak, jak putuje po datovém nosiči, zjistíme, že k datům, které jsme vložili do počítače byla přibalena ještě další data a to dokonce několikrát po sobě.

Síťová komunikace je tedy rozdělena do několika vrstev, kde se každá vrstva stará o pevně danou část komunikace. Ke své činnosti využívá *rozhraní* sousedních vrstev a se svými vrstevníky komunikuje pomocí *protokolu*. Existuje několik druhů síťových zařízení, jednoduše proto, že existuje několik různých síťových vrstev. Každé zařízení se stará o příslušnou vrstvu, pouze vyšší datové vrstvy zpravidla zpracovává nějaký software běžící na procesoru, neboť protokolů těchto vrstev je nepřeberné množství. Tímto způsobem se zpracování paketu rozdělí mezi různé funkční bloky, čímž se dosáhne jednak lepší spravovatelnosti a jednak vyšší výkonnosti.

I když existuje referenční ISO/OSI model vrstev, pro naší potřebu poslední tři vrstvy (aplikační, prezetační a relační) spojíme do jediné. Úplný popis lze najít v [2, s.69].

1. *Fyzická vrstva* má na starost fyzické spoje. Definuje pro ně elektrické a fyzikální vlastnosti, stanovuje způsob přenosu dat. Konvertuje digitální signály na takové signály, které používá přenosové médium.
   Na této vrstvě pracují *rozbočovače*, *opakovače* a *síťové adaptéry*.

2. *Linková vrstva* zajišťuje přístup ke sdílenému médiu a adresaci na fyzickém spojení, tj. v jednom fyzickém segmentu. Vytváří rámce z dat, přidává jim fyzickou adresu (na Ethernetu je to MAC adresa). Také se stará o nastavení parametrů spoje a kontrolu chyb při přenosu.
   Tuto vrstvu zpracovávají *mosty* a *přepínače*.

3. *Síťová vrstva* má za úkol směrovat pakety sítí. Používá k tomu logické adresy. Spojuje spolu zařízení, které spolu přímo nesousedí. Nejznámější protokol této vrstvy je *IP*.
   Zpracovávání této vrstvy zajišťují *routery*.

4. *Transportní vrstva* zajišťuje přenos dat mezi koncovými body. Poskytuje především kvalitu přenosu, kterou požadují protokoly vyšší vrstvy. Principielně se její protokoly dělí do dvou skupin: spojově orientované a nespojově orientované služby. Zpracování této a vyšších vrstev zpravidla probíhá v softwaru konkrétního systému.
   Protokoly této vrstvy jsou především *TCP* a *UDP*.

5. *Aplikační vrstvu* již implementuje každá aplikace jinak, podle své potřeby.
   Řadí se sem například protokoly *HTTP*, *SSH* nebo *DHCP*.

2.2 Routery a jejich funkce

Jak již bylo řečeno, router je zařízení pracující na síťové vrstvě. Spojuje různé sítě tak, že na základě protokolu IP (podle cílové IP adresy) směruje pakety do příslušných sítí, tzn. na konkrétní rozhraní.

Jako router můžeme obecně použít i běžný počítač s příslušným softwarem. To ale dostačuje jen pro malé sítě, pro provoz ve vysokorychlostních a rozsáhlých sítích je nutné použít specializovaný hardware. Dnes už jsou cenově uspokojivé i dedikované routery pro malé sítě. Většinou se jedná o sítě pro domácnosti nebo malé kanceláře, kde chceme několik počítačů propojit mezi sebou a zároveň jim poskytnout přístup k internetu.

Tato zařízení většinou uživatelům nabízí víc, než jen routování. V podstatě je na zařízení spuštěný operační systém, do kterého už je snadné modulárně přidávat další potřebnou funkcionalitu. Pojďme se podívat, co běžné domáci routery nabízí.

2.2.1 Směrování

Směrování je základní funkce routeru. Při tomto procesu se každý přijatý paket z nějakého síťového rozhraní odešle na jiné síťové rozhraní. O jaké odchozí rozhraní se jedná, router pozná podle cílové IP adresy paketu. Projde tabulku se známými adresami síťí (tato tabulka může obsahovat i konkrétní IP adresy) a zjistí pro jakou síť je paket určen. V tabulce je také uložena identifikace rozhraní, na kterém daná síť leží. A právě toto rozhraní bude použito pro odeslání paketu.

V domácích routerech běžně bývají dvě rozhraní, na které je možné směrovat pakety. Jedno je připojeno k internetu, druhé do vnitřní sítě. Bývá téměř pravidlem, že druhé rozhraní je ještě v routeru rozděleno na několik dalších pomocí přepínače. Pro uživatele je to většinou výhoda, protože nemusí kupovat rozbočovač a tak odpadá i nutnost spravovat další zařízení.

Lze tedy říci, že směrování na domácích routerech je jednoduchý proces přeposílání paketů z lokální sítě do internetu a naopak.

2.2.2 DHCP

DHCP je služba, která má na starosti automatickou konfiguraci zařízení v síti. Na nějakém zařízení v síti (tedy v našem případě na routeru) je spuštěn DHCP server přijímající požadavky od klientů, kteří (zpravidla při spuštění systému) chtějí získat všechny údaje k tomu, aby mohli komunikovat s okolím. Mezi tyto údaje patří IP adresa, maska sítě, IP adresa výchozí brány a adresy DNS serverů pro internet. V routeru je většinou možnost konfigurovat všechny uvedené parametry.

2.2.3 NAT

Funkce NAT, neboli překlad síťových adres (Network Adress Translation) byla navržena pro možnost připojit několik počítačů do internetové sítě. Kromě tohoto způsobu připojení máme ještě možnost použít přímé mapování několika IP adres, které nám může zapůjčit poskytovatel internetového připojení. Pokud ale nemáme to štěstí a nemůžeme se poskytovatelem dohodnout na propůčení více adres (za každou další adresu většinou zaplatíme), zbývá maskovat adresy z vnitřní sítě za jednu adresu z vnější sítě.

Většina internetových protokolů funguje nad protokolem TCP, který používa čísla portů (z rozsahu 0 - 65 535) především na rozlišování služeb vyšších vrstev. Běžný počítač najednou nekomunikuje na více než 100 portech najednou, takže v domácích sítích lze jednoduše aplikovat následující postup:

1. V odchozím požadavku (tj. v paketu, který směřuje z vnitřní sítě do vnější, internetové) přepíšeme zdrojovou adresu na adresu přidělenou poskytovatelem.

2. Přepíšeme přitom také zdrojový port na nějaký port ze seznamu nepoužitých portů. Obě tyto hodnoty spolu s původní zdrojovou adresou si poznačíme do tabulky.

3. Takto upravený paket odešleme do vnější sítě.

4. Jakmile přijde odpověď zpět, vyhledáme v tabulce záznam podle čísla portu a podle těchto hodnot přepíšeme IP adresu a port na původní hodnotu.
   Pokud nenajdeme v tabulce záznam s číslem portu, příchozí paket zahodíme (popřípadě předáme další funkci, která paket může zpracovat jiným způsobem). Tímto způsobem tedy máme navíc zajištěnou i určitou bezpečnost.

5. Modifikovaný paket odešleme na původní zdrojovou adresu do vnitřní sítě.

NAT ke své funkci nepotřebuje žádné nastavení, u některých routerů se dokonce ani tato funkce nedá vypnout.

2.2.4 Firewall

Firewall je funkce (nebo dokonce samostatné zařízení implementující tuto funkci), která slouží k zabezpečování síťového provozu. Slouží jako bod, který definuje pravidla pro komunikace mezi různými sítěmi, které dělí. Pravidla zde většinou znamenají omezení nebo povolení přístupu k síťovým službám.

Rozsah implementace firewallu v routeru se většinou silně liší, už jen proto že existuje více druhů firewallu.

Paketové filtry

Nejjednoduší příklad firewallu spočívá v tom, že pravidla určují, jaké adresy a jaké porty jsou povolené. Pakety z těchto adres a s těmito porty pak mohou být propuštěny do sítě na druhé straně firewallu. Je tedy kontrolována síťová a transportní vrstva, podle modelu ISO/OSI.

Výhodou této metody je rychlost zpracování paketů, nevýhodou naopak nízká možnost kontroly průchozích paketů. To u složitějších protokolů (např. FTP, RPC) dokonce zabraňuje správnému fungování příslušné služby.

Typickým zastupcem paketových filtrů jsou algoritmy založené na ACL (Access Control List, doslova seznam k řízení přístupu).

Aplikační brány

Aplikační brány (též známe pod názvem Proxy firewally), narozdíl od předchozího typu, úplně oddělují sítě, mezi kterými stojí. Nepropouští žádné pakety, namísto toho je analyzují a vytvoří jiné s podobným obsahem. Klienti tím pádem s požadovaným serverem nevytváří spojení přímo, ale vytvoří spojení s firewallem a až ten na základě požadavku vytvoří spojení s cílovým serverem. Zjednodušeně lze říci, že je to NAT, který provádí překlad dat i v aplikační vrstvě, ovšem pro síťovou komunikaci není transparentní.

Ačkoliv tento typ řešení firewalu poskytuje velmi dobré zabezpečení, v porovnání s paketovým filtrem je schopný zpracovat pakety mnohonásobně nižší rychlostí a má také mnohem vyšší latenci. Navíc na každý protokol je potřeba vytvořit specializovaný modul (lze použít tzv. generickou proxy, ta však neposkytuje o mnoho lepší bezpečnost než paketový filtr).

Stavové paketové filtry

Tyto filtry kontrolují data podobně jako jednoduché paketové filtry, navíc si ale ukládají informace o povolených spojeních. Ty pak využívají při rozhodnutí, zda příchozí paket spadá do povoleného spojení. Pokud ano, mohou být přeposlány dál k cílovému počítači, v opačném případě jsou podrobeny testům na klasických pravidlech.

Je vidět, že tento mechanizmus urychluje například zpracování paketů již otevřencých spojení. Navíc v pravidlech pro firewall stačí uvádět jen směr pro navázání spojení, změnší se tedy významně počet pravidel. Stavové paketové filtry nabízejí poměrně slušnou úroveň zabezpečení. V operačním systému Linux je najdeme jako volně dostupný nástroj *iptables*.

2.2.5 Port Forwarding

Pokud potřebujeme přistupovat z vnější sítě ke službám nějakého zařízení a nemáme dostatek veřejných IP adres, můžeme využít funkce přeposílání určitého portu. Na routeru se nastaví čísla portu, pro které se příchozí pakety nebudou ignorovat, místo toho se přepošlou na specifickou vnitřní IP adresu. Také tento port nebude k dispozici pro NAT.

Zjednodušeně lze říci, že Port Forwarding pracuje podobně jako NAT, jen s opačným směrem spojení.

2.2.6 VLAN

Virtuální síť LAN (local area network) je skupina počítačů, které se chovají tak, jako by byly připojeny do jedné fyzické sítě (např. přes switch), nezávisle na jejich fyzickém umístění. Lze tedy mít počítače kdekoliv na světě a přitom mezi sebou mohou komunikovat, jako v jedné broadcastové doméně[1]. Využívá se při tom několika počítačových sítí (i internetu), které musí být vzájemně propojené. Pakety pro VLAN mohou putovat celou touto sítí.

[1]Skupina počítačů takových, které přijímají broadcastové pakety od jakéhokoliv zařízení v této skupině. Broadcastovou doménu většinou ohraničuje router, protože ten nepřeposílá broadcastové rámce.

2.3 Routery vybrané pro semestrální projekt

V procesu výběru routerů zohledněny následující požadavky:

- Podpora operačního systému OpenWRT
- Různá architektura procesoru (ARM, MIPS)
- 100 Mbit nebo 1 Gbit rozhraní Ethernet
- Dostupnost za přiměřenou cenu na trhu v České Republice

Pro měření výkonnosti byly určeny následující zařízení:

Název	Procesor, frekvence [MHz]	RAM / ROM [MiB]	Ethernet
Linksys WAG 160N	MIPS, BCM6538 300	32 / 4	4x 100 Mb + ADSL
D-Link DIR-825	MIPS, AR7161 680	64 / 8	4x 1Gb + 1x 1Gb
Avila GW2348	XScale, IXP425 533	64 / 16	2x 100 Mb
Seagate Dockstar	ARM, Kirkwood 1200	128 / 256	1x 1Gb

2.3.1 Linksys WAG 160N

Tento router je zároveň modemem pro ADSL. Obsahuje Wi-Fi modul, takže může být použit i jako přístupový bod pro bezdrátou síť. Najdeme u něho funkce jako NAT, Firewall, blokování konkrétních adres a další. V routeru je vestavěný 100 Mbit Ethernet a obsahuje proprietární operační systém.

2.3.2 D-Link DIR-825

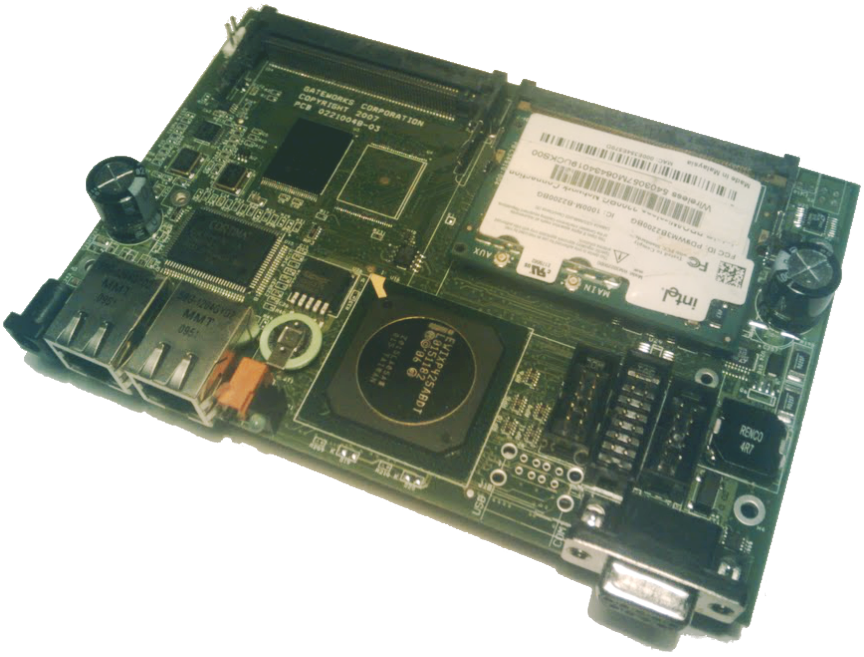
Toto zařízení je router s gigabitovým Ethernetem. Také obsahuje Wi-Fi modul, dokonce s podporou 2.4 GHz a 5 GHz nosného signálu. Dále zde najdeme USB konektor (vevnitř zařízení je schovaný ještě jeden) a jako software opět proprietární operační systém.

2.3.3 Avila GW2348

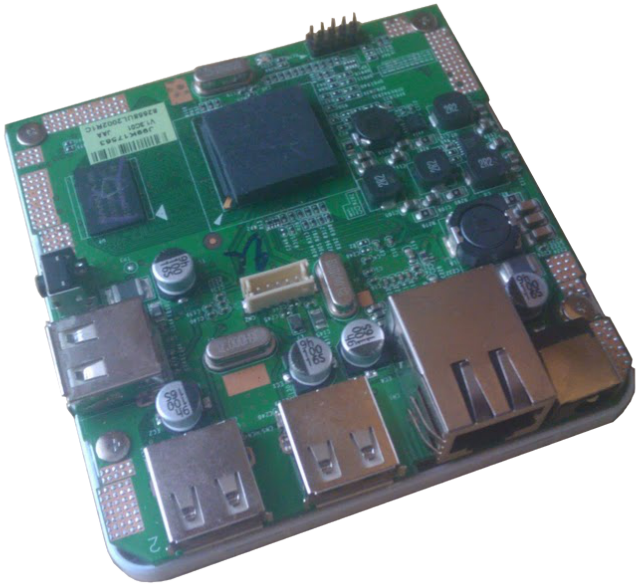
Gateworks Avila je vývojová deska pro síťové aplikace. Obsahuje XScale procesor založený na jádře ARM. Na desce najdeme spoustu periferií: 4 sloty Mini-PCI, např. pro Wi-Fi moduly, dva seriové porty, GPIO, Compact Flash slot, USB rozhraní. Deska podporuje vývoj na systému Linux a je dodávána s operačním systémem OpenWRT. Jak vypadá se můžeme podívat na obrázku 2.1.

2.3.4 Seagate Dockstar

Seagate Dockstar není router, ale dokovací stanice pro USB disk, tedy NAS. Vybrali jsme ho však z toho důvodu, že se nám nepodařilo se nám najít klasický router s procesorem ARM. Obsahuje jedno 1 GbE rozhraní a celkem 4 USB konektory. V zařízení je dodáván upravený systém PogoPlug. Vnitřek zařízení je zachycen na fotografii 2.2.



Obrázek 2.1: Vývojová platforma Gateworks Avila



Obrázek 2.2: Dokovací stanice Seagate Dockstar bez vrchního krytu

Kapitola 3

Architektura ARM

Architektura procesorů ARM byla navržena tak, aby její hardwarové implementace mohly být co nejmenší a nejvýkonnější. Právě díky její relativní jednoduchostí, takže i nízké spotřebě, si našla velké uplatnění na trhu s mobilní elektronikou a vestavěnými systémy.

ARM je procesor typu RISC, od toho také dědí hlavní vlastnosti těchto procesorů:

- **Architektura Load/Store**. Operace s daty jsou možné jen nad registry, tedy ne nad daty ve hlavní paměti.
- **Sada registrů**. Je zde celkem 31 registrů k obecnému použití, v uživatelkém režimu je možné používat jen 16. Navíc poslední dva jsou *Link Register* a *Program Counter*.
- **Konstantní délka kódu instrukce**, vede ke zjednodušení dekódování instrukce.

Naopak tyto vlastnosti byly v návrhu ARM zamítnuty:

- **Registrová okénka**, např. pro vnořené funkce. Hlavní problém je zde kvůli velkému prostoru na čipu, který by musel být rezervován pro spoustu registrů. Na druhou stranu každý režim procesoru má vlastní sadu registrů, což je podobný princip.
- **Zpožděné skoky**. Samotné skoky způsobují nepříjemnosti pro zřetězené linky, protože přeruší posloupnost instrukcí. Většinou se tento problém řeší tak, že se skok provede až za následující instrukcí. Jenže v tom případě to ruší atomicitu prováděných instrukcí.
- **Všechny instrukce se vykonají za jeden hodinový takt**. I když ARM vykonává většinu instrukcí v jednom taktu, spousta dalších (např. Load/Store instrukce) potřebují dva přístupy do paměti (pro instrukci a pro data).

3.1 Obecné informace o procesorech

3.1.1 Instrukční sada

ARM disponuje dvěmi sadami instrukcí: klasický 32-bitový formát a komprimovaný 16-bitový Thumb. My se budeme dále zabývat pouze 32-bitovu sadou. Instrukce ARM mají tyto vlastnosti:

- Load-Store architektura
- Výpočetní instrukce adresují data třemi adresami (dva zdrojové a jeden cílový)
- Podmínečné provedení každé instrukce
- Instrukce pro uložení / načtení několika registrů najednou.
- Možnost provést operaci bitového posunu a nějakou operaci ALU v jednom hodinovém taktu: `a += (j << 2);`

3.1.2 Zřetězené linky

Zřetězená linka (neboli pipeline) je soustava několika výpočetních částí, kde se data výstupu nějaké části vkládají do části následující v dalším hodinovém taktu. Každá část procesoru tedy může zpracovávat jinou instrukci, resp. její část. Několik prvních verzí jádra ARM měly třístupňovou zřetěženou linku:

1. **Fetch** - instrukce je načtena z paměti a vložena do instrukční linky.
2. **Decode** - instrukce je dekódována, jsou připraveny řídící signály.
3. **Execute** - hodnoty jsou vyčteny z registrů, ALU vygeneruje výsledek a ten je zapsán zpět do některého z registrů.

V pozdějších verzích jader (přibližně od r. 1995) přibyly další dva stupně:

4. **Buffer/data** - přístup do paměti, pokud je potřeba. V opačném případě je výsledek z přechozího kroku ALU (Execute) jednoduše pozdržen.
5. **Write-back** - výsledek provedení instrukce je zapsán zpět do příslušného registru.

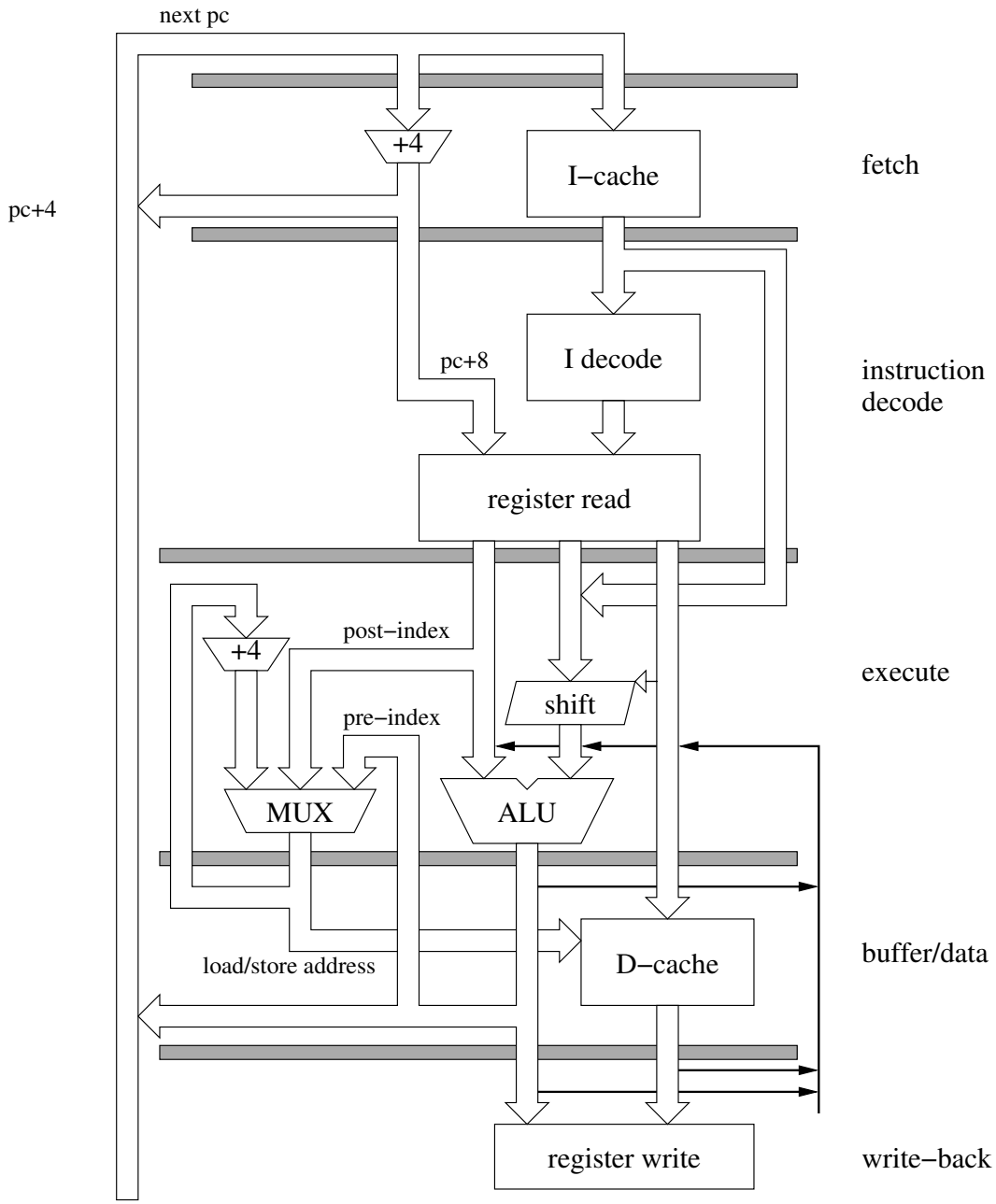
Takovouto architekturu můžeme vidět na obrázku 3.1. Ovšem těchto pět stupňů není konečný počet, v nejnovějších jádrech můžeme vidět například až třináct stupňů (Cortex-A8).

3.1.3 Další zajímavosti

Společnost ARM Ltd. procesory přímo nevyrábí, pouze vytváří jejich specifikace, nástroje pro vývoj a hlavně vyvíjí jádro, které pak licencuje jiným společnostem.

Aby bylo možné ladit aplikace bez softwarových nástrojů (třeba kód zavaděče), je ve většině procesorů ARM k dispozici rozhraní JTAG a s ním i EmbeddedICE, což je dnes víceméně standard pro ladění.

Některé procesory mají jednotku, která dovoluje přímo provádět bytový kód Javy. Zrychlí tak provádění některých aplikací na mobilních telefonech.



Obrázek 3.1: Architektura jádra ARM s 5-ti stupňovou linkou. Převzato z [10].

3.2 Marvell Kirkwood

Tento procesor obsahuje jádro Sheeva, které plně vyhovuje specifikaci ARMv5TE, nepodporuje tedy hardwarové zpracování operací v plovoucí řádové čárce. Jádru je k dispozici 16 + 16 KiB instrukční + datové cache L1 a 256 KiB L2 cache. Dále obsahuje například jednotku pro správu paměti (MMU), jednotku predikce skoků a 64 b interní datovou sběrnici.

Procesor poskytuje rozhraní PCI Express x1, SATA II, USB 2.0 a spoustu dalších rozhraní s menší přenosovou kapacitou (I2C, SPI). Jsou zde vestavěny hardwarové jednotky pro šifrování a autentizaci dat pomocí algoritmů AES, DES a 3DES a výpočet kontrolních součtů pomocí SHA1 a MD5.

Rovněž logická funkce XOR je akcelerovatelná hardwarově, to lze využít například při výpočtu CRC a podobně. Tato XOR jednotka obsahuje čtyři nezávisle konfigurovatelné bloky, které mohou fungovat jako CRC, XOR, DMA, nebo Memory Init. Této jednotky se využívá například v linuxovém jádře, pro akceleraci funkcí memset, memcpy a podobně.

3.3 Intel IXP425

V tomto procesoru je vestaveno jádro XScale (založeno taktéž na ARMv5TE). K jádru patří 32 + 32 KiB instrukční + datové cache L2, zároveň má ještě 2 KiB datové cache L1. Jádro používá sedmistupňovou zřetězenou jednotku pro celočíselné operace a osmistupňovou jednotku pro paměťové operace. Je to nástupce procesorů řady StrongARM (s jádrem ARMv4).

Podstatnou vlastností tohoto síťového procesoru jsou tři vestavěná akcelerační jádra NPE (network processing engines). Tato jádra pracují nezávisle na hlavním procesoru a mohou akcelerovat síťový provoz několika různými mikroprogramy. Bohužel firma Intel k těmto jádrům neposkytuje žádnou dokumentaci, takže nelze vytvářet nové mikroprogramy. Jsme omezeni pouze na režimy Ethernet, ATM, Crypto, High-Speed Serial a některé jejich kombinace. Každé jádro navíc může provádět jen některé režimy.V praxi jsou tedy dvě jednotky použity pro zpracování linkové vrstvy Ethernetových sítí a třetí zůstává nevyužita.

Bohužel, ani u procesororu Marvell Kirkwood, ani u Intel IXP425 nebylo možné získat dokumenty popisující přesnou strukturu jádra. Ty jsou dostupné pouze pro spolupracovníky s povinností podepsat NDA. Přesto by tyto dokumenty byly užitečné, protože jak uvidíme, při testování výkonnosti jsme došli k zajímavým a někdy velice rozdílným výsledkům

Kapitola 4

Algoritmy používané v síťových aplikacích

4.1 Hledání nejdelšího shodného prefixu (LPM)

Od síťových směrovačů je očekáváno rychlé přeposlání paketů (Fast Packet Forwarding) na správné rozhraní. Je možné k tomu použít jednu ze dvou technik.

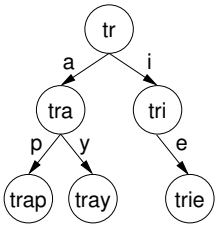
První z nich přidává speciální hlavičky k paketu. Přeposlat paket na základě rozhodnutí učiněného podle této hlavičku s adresou, která je menší než IP adresa, trvá jednoznačně kratší dobu než druhý uvedený princip. Nicméně velká nevýhoda je nutnost přidat k paketu další informaci. V klasických sítích se v podstatě nepoužívá, avšak na poobném principu pracuje například protokol MPLS. Ten ale neslouží k směrováním paketů podle IP adresy, ale spíše k vytvoření jakýchsi virtuálních sítí.

Druhou technikou technikou jsou právě metody hledání nejdelšího společného prefixu, ukážeme si dva tyto algoritmy - Tree Bitmap (TBM) a Shape Shifting Trie (SST). Tyto metody vyhledávají cílové rozhraní ve směrovací tabulce. Nejpoužívanějším prostředkem pro vytvoření takovéto tabulky je protokol BGP.

4.1.1 Trie

Trie se velice často používá v algoritmech provádějících LPM. Je tedy důležité o ní vědět alespoň základní informace.

Trie, nebo také prefixový strom, je datová struktura používaná pro uchování hodnot asociativního pole a přístup k nim. Klíči pro přístup k těmto položkám většinou bývá nějaký řetězec. Trie se podobá binárnímu stromu s jedním velkým rozdílem. Uzel v binárním stromu hodnotu klíče přímo obsahuje ale v trie je hodnota klíče určena pozicí ve stromové hierarchii. Pro lepší představu uveďme příklad trie na obrázku 4.1.



Obrázek 4.1: Příklad jednoduché trie

Pro ukázku je zde v pseudokódu zapsán algoritmus nalezení jedné položky:

```
function find(node, key)              // Funkci se nejčastěji předává
begin                                 // kořenový uzel
    for char in key                   // Postupujeme po znacích
    begin
        if char in node.childs        // Pokud je aktuální znak v některém
            node = node.childs[char]  // ze synů, přesuň se na tohoto syna
        else
            return None               // Jinak položka v trie neexistuje
    end
    return node.value
end.
```

4.1.2 Algoritmus Tree Bitmap

V této sekci popíšeme základní myšlenku algoritmu zvaného Tree Bitmap[3]. Je to algoritmus pracující s více bity trie najednou, dokáže vyhledávat ve stromu prefixů rychleji než jednobitové přístupy.

Díky parametrizovatelné velikosti multi-uzlu pak lze optimalizovat velikost uzlu v paměti například tak, aby byl získán v jednom přístupu do paměti (předpokládáme-li proudový režim pamětí, tzv. burst, ve kterém je při jednom přístupu do RAM dodán větší blok paměti, např. 32 bytů).

První idea v algoritmu Tree Bitmap je ta, že všechny synovské uzly nějakého uzlu trie jsou uloženy přímo za sebou. Takto je možné použít pouze jeden ukazatel pro všechny synovské uzly (tyto ukazují na začátek bloku synovských uzlů), protože pozice každého syna může být vypočítána jako nějaký offset z jednoho ukazatele. To také sníží počet potřebných ukazatelů v porovnání se standartními vícebitovými triemi na polovinu a tím také paměťovou náročnost uzlu.

Za druhé jsou v každém uzlu použity dvě bitmapy, které obsahují hodnoty 1 na pozicích, kde je definovaný nějaký prefix. Jedna je zde pro všechny prefixy uložené uvnitř a druhá pro prefixy uložené mimo, v dalších uzlech. Z toho vyplývá, že bitmapa pro vnitřní prefixy obsahuje 2^{r-1} položek, pro vnější 2^r, kde r je výška stromu v multi-uzlu.

Třetí nápad je zachovat uzly trie co nejmenší, abychom redukovali počet přístupných přístupů do paměti pro konkrétní délku stromu multi-uzlu. Proto je multi-uzel konstatní velikosti, obsahuje pouze ukazatel na bitmapu externích ukazatelů, bitmapu interních uzlů a jeden ukazatel na blok synovských uzlů.

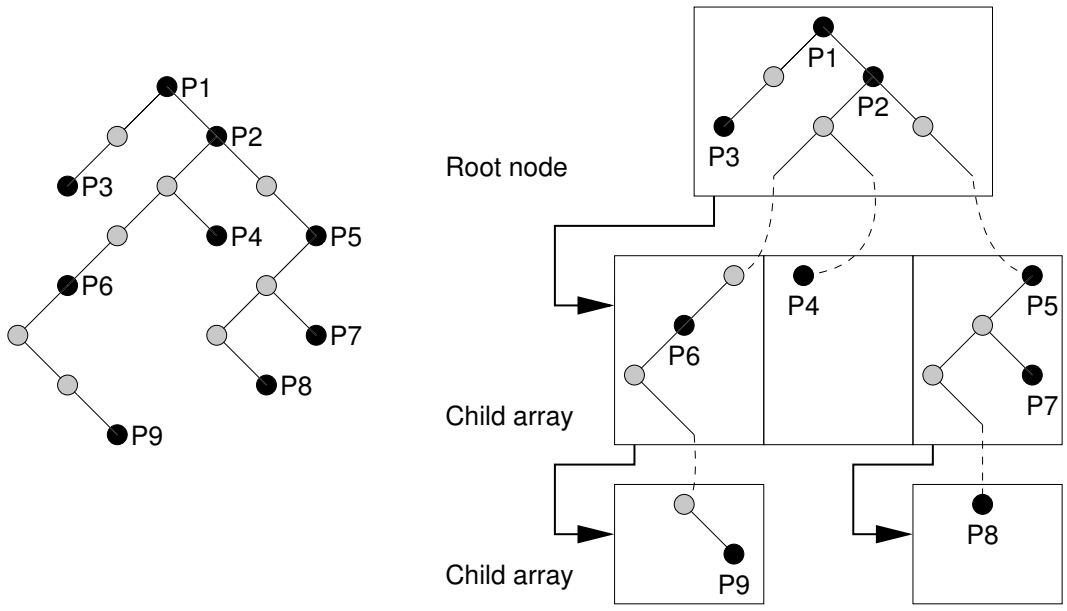
Na obrázku 4.2 vlevo je zobrazena nějaká trie. Tmavě vybarvené jsou uzly, pro které existuje hodnota, pro světlé uzly hodnota není definovaná. Vpravo je pak odpovídající struktura bloků multi-uzlů tak, jak ji dokáže zpracovat algoritmus Tree Bitmap.

V pseudokódu by pak funkce pro vyhledávání pomocí Tree Bitmap mohla vypadat přibližně takto:

```
begin
    node := root;   // node je aktuální uzel trie, který je prohledáván.
                    // Začínáme kořenovým uzlem.
    i := 1;         // i je index do pole částí hledané hodnoty (stride).
                    // Začínáme první částí (tzn. bity [0..n/r])
    while (1)
        // Pokud je zde nejdelší shodný prefix, aktualizuj ukazatel.
        if (findLongestMatch(node.intBitmap, stride[i]) != null )
            LongestMatch := node.ResultsPointer + CountOnes(
               node.intBitmap, treeFunction(node.intBitmap, stride[i]));

        // Pokud neexistuje žádná cesta do dalšího externího uzlu,
        // výsledek vyhledávání je poslední zaznamenaný interní uzel.
        if (node.extBitmap[stride[i]] = 0)
            return Result[LongestMatch];

        // Cesta existuje, přesuneme se na další uzel.
        else
            node:= node.childPointer + CountOnes(node.extBitmap, stride[i]);
        i := i+1;
    end while;
end.
```



Obrázek 4.2: Příklad trie a odpovídající struktuře pro algoritmus Tree Bitmap

4.1.3 Algoritmus Shape Shift Trie

Podobně jako algoritmus TBM, i SST vychází z konceptu binární trie. Můžeme říci, že se dokonce jedná o zobecněnou variantu TBM. Opět jsou zde uzly trie spojené do multi-uzlů. Zde však trie v multi-uzlu není úplný vyvážený strom, ale tvar trie může být zcela libovolný. Omezením je pouze maximální počet uzlů v jednom multi-uzlu.

Algoritmus má o něco menší paměťovou náročnost vůči TBM při řídkých prefixových sadách, díky možnosti optimalizovat každý multi-uzel na přesný tvar. Bohužel toto je zároveň nevýhodou, protože se ve výpočtu musí uvažovat i s tvarem uzlu. Pro tento algoritmus tedy přibyla v datové struktuře položka popisující tuto vlastnost uzlu.

Narozdíl od TBM, díky možnosti výběru tvaru trie pro multi-uzel zde může existovat několik výsledných trie pro jednu vstupní. Efektivita algoritmu pak tedy závisí i na konkrétním trie. Jak lze vytvořit optimální SST trie můžeme zjistit například v [11], kde je také podrobnější popis tohoto algoritmu.

4.2 Algoritmy pracující s Bloomovy filtry

4.2.1 Základní Bloomův filtr

Bloomův filtr je datová struktura, která ukládá nějakou množinu dat. Tato datová struktura je určena polem bitů o velikosti m a k hashovacími funkcemi. Velikost pole bitů lze snadno parametrizovat, stejně tak konkrétní hashovací funkce můžeme vybírat libovolně, jen je zapotřebí aby jejich výstupní hodnoty byly v rozsahu 0-m Nad touto strukturou jsou definovány tři operace:

1. Inicializace Bloomova filtru – všech `m` bitů bitového pole se nastaví na 0.
2. Vložení prvku do Bloomova filtru – pro vkládaný prvek se vypočítá hash a to hned několika různými hashovacími funkcemi. Každý výstup z hashovací funkce je potom ukazatel do bitového pole Bloomova filtru. Bitové pole se tedy indexuje pomocí těchto hashí a na příslušná místa zapíší bity s hodnotou 1.
3. Dotaz na existenci prvku v Bloomově filtru – pro prvek, u něhož chceme znát jeho příslušnost do množiny reprezentované Bloomovým filtrem, opět vypočítáme všechny hashe. Pokud se na všech indexech bitového pole, daných těmito hashemi, vyskytují 1, prvek v Bloomově filtru existuje. V opačném případě, pokud alespoň na jednom indexu bitového pole je zapsána 0, prvek v Bloomově filtru neexistuje.

Jak je vidět, neexistují zde operace pro odstranění prvku ani pro získání hodnoty prvku. Z principu algoritmu také plyne další zásadní vlastnost a to, že výsledek dotazu na existenci prvku může být nepravdivý. To může být důsledek toho, že několik vložených prvků způsobí postupně zapsání 1 do všech takových bitů pole, jejichž indexy odpovídají indexům hashí tázaného prvku.

V tomto případě pak označujeme dotaz na prvek v Bloomově filtru za False Positive. Pravděpodobnost tohoto jevu lze vyjádřit vztahem:

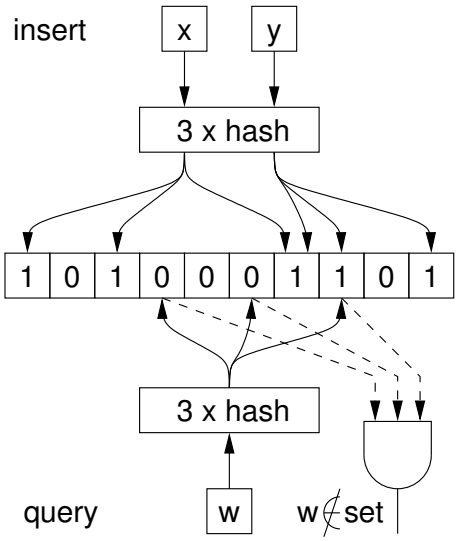
$$f = (1 - e^{\frac{-nk}{m}})^k$$

kde m je počet bitů pole, k je počet hashovacích funkcí a n je počet vložených prvků.

4.3 Čítající Bloomův filtr

Čítací Bloomův filtr je modifikace původního konceptu, která umožňuje navíc položky z filtru odebírat. Místo m-bitového pole zde je pole m p-bitových čítačů.

Jelikož čítače mají omezený rozsah, správné přidání prvku (tak, aby ho bylo možné i odebrat) je závislé na počtu vzniklých kolizí. Tzn. pokud se na jeden čítač namapuje více prvků, než je jeho rozsah, při naplnění zůstane nastaven na své nejvyšší hodnotě. (Pokud by čítače přetíkaly, vznikly by falešné negativní výsledky.) Následně prvky, které zasahují do takovéhoto čítače, už není možné odebrat, protože by při tom čítač klesnul na hodnotu 0 před odebráním všech prvků.



Obrázek 4.3: Princip základního Bloomova filtru

4.4 Algoritmy pro vyhledávání vzorů (Pattern Match)

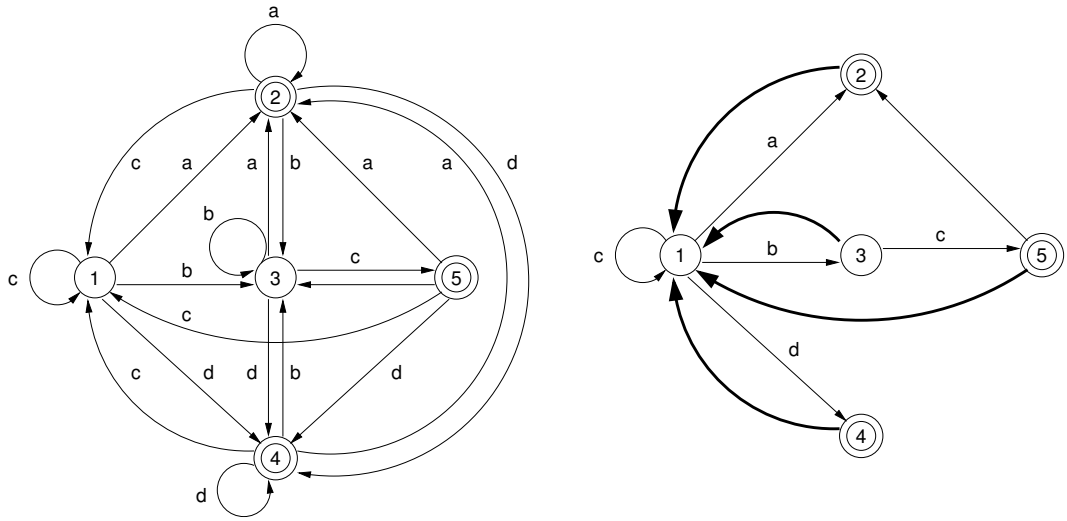
4.4.1 Algoritmus Delay DFA

Je známé, že každý regulární výraz lze popsat konečným automatem. Velikost automatu je jednak dána počtem stavů, jednak počtem přechodů mezi stavy. Pro abecedu složenou z ASCII znaků může teoreticky z jednoho stavu vést až 256 přechodů. Složitější regulární výraz tak může reprezentovat rozsáhlý konečný automat, který může v paměti počítače zabírat velký prostor.

Představme si tedy automat, který může obsahovat speciální přechody, nazvěme je implicitní. Implicitní přechod se provede vždy, když není k dispozici přechod pro aktuálně zpracovávaný znak. Samozřejmě se tento znak musí vždy zpracovat, takže automat přechází mezi přechody, dokud neprovede přechod příslušící tomuto znaku.

V [9] byl tento algoritmus předveden a bylo ukázáno, že pro složité regulární výrazy redukuje počet hran automatu až o 95%. Bylo také ukázáno, že převod deterministkého automatu na Delay DFA je NP-těžký problém.

Jak vypadá jednoduchý Delay DFA automat můžeme vidět na obrázku 4.4.

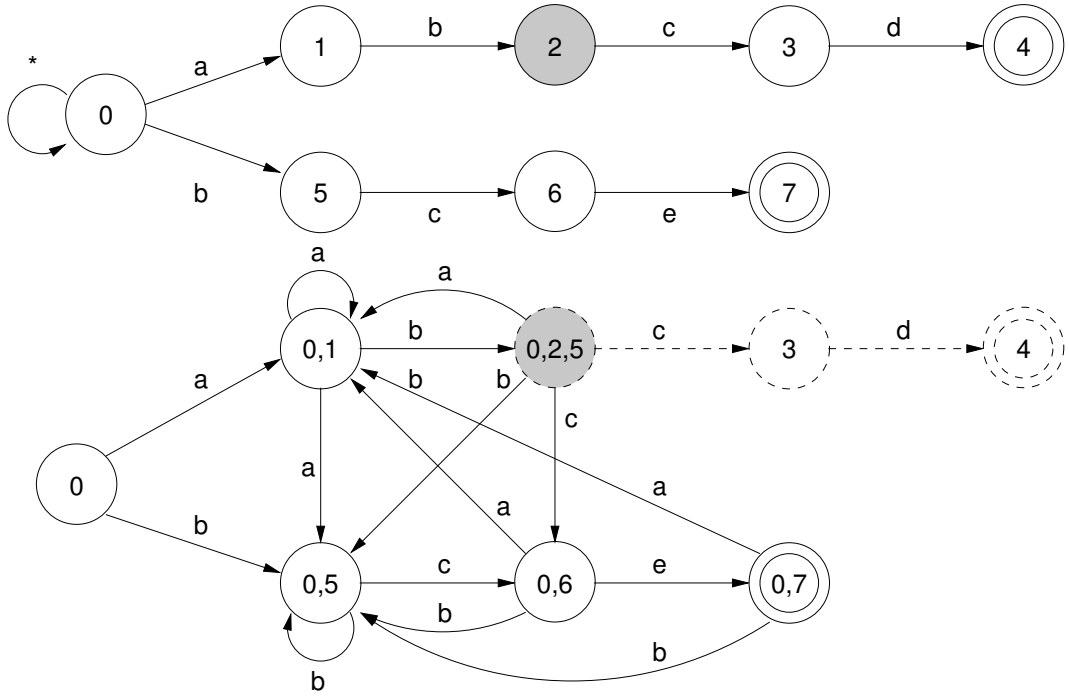


Obrázek 4.4: Vlevo je znázorněn konečný automat přijímající a^+,b^c a c^*d^+, vpravo ekvivalentní Delay DFA s implicitními přechody vyznačenými tučně. Převzato z [9].

4.4.2 Algoritmus Hybrid FA

I tento algoritmus se snaží vypořádat s paměťovou náročností konečného automatu. Konkréně je použitelný v případech, kdy potřebujeme složit dva regulární výrazy do jednoho automatu. Totiž, pokud sloučíme dva konečné automaty reprezentující dva regulární výrazy do jednoho automatu, počet stavů tohoto nového automatu nebude součtem počtu stavů dílčích automatů, ale bude růst exponenciálně.

Jednou metodou, která řeší tento problém je právě algoritmus Hybrid FA popsaný v [8]. Ten kombinuje přístupy deterministických a nedeterministických automatů: Pokud se v nějakém kroku převodu NFA na DFA ukáže, že při něm dochází k značnému nárůstu počtu stavů, proces se přeruší a tato část automatu zůstane nedeterministická. Obrázek 4.5 názorně zobrazuje podstatnou část tohoto problému.



Obrázek 4.5: Nahoře je znázorněn nedeterministický konečný automat přijímající *abcd* a *bce*. Pod ním je ekvivalentní Hybrid FA, čárkovaně je naznačená nedeterministická část automatu. Převzato z [8].

Kapitola 5

OpenWRT

OpenWRT je linuxová distribuce, kterou najdeme především na vestavěných zařízeních a to konkrétně na takových, které mají co dočinění s počítačovou síťí. Je postavena na Linuxovém jádře s drobnými úpravami a lze v ní najít spoustu softwarových balíčků (ty jsou spravovány pomocí nástroje opkg). OpenWRT je možné konfigurovat přes příkazový řádek, tak, jak je to pro linuxový systém přirozené, nebo pomocí modulárního webového rozhraní, které se nazývá LuCI.

Projekt vznikl po uvolnění firmware routeru WRT54G od firmy Linksys. Ta si od toho slibovala vývoj distribuce pomocí komunity a to se jí také podařilo. Postupně distribuce začala podporovat více a více zařízení.

5.1 Klíčové vlastnosti distribuce

- Rošířené možnosti konfigurací sítí, možnost použití VLAN s mnoha možnostmi konfigurace pro vlastní proces směrování
- Možnosti filtrovat, manipulovat a zpožďovat pakety
- Funkce Firewallu
- Forwardování portů vnějšího síťového provozu počítačům uvnitř sítě, které jsou schovány za NAT
- Uplatnění kvality služeb (QoS) pro všemožné služby jako VoIP, online hry a multimédiální přenosy
- Omezování provozu uživatelů pro spravedlivé rozdělení pásma
- Loadbalancing - rozdělování vytížení linky mezi více poskytovatelů internetu.
- Tunelování IP provozu
- Monitorování a statistiky provozu sítě v reálném čase
- Statické IP adresy pro DHCP
- UPnP technologie pro dynamické nastavení forwardování portu
- Příjemné webové rozhraní založené na technologii AJAX, které se nazývá LuCI.

- Pro zařízení, které obsahují USB port je možnost:
  - Sdílení tiskárny,
  - Sdílení souborů pomocí protokolu SAMBA (tento protokol nativně podporuje systém Windows)
  - Přehrávání zvuku za pomoci USB zvukové karty
  - Funkce webové kamery
  - Další zařízení, které mají podporu v linuxovém jádře

5.2 Instalace systému OpenWRT do zařízení

Obecně je postup nahrání jiného firmwaru do zařízení poměrně složitý a zdlouhavý proces. Navíc hodně závisí na znalosti architektury jak zařízení tak instalovaného systému. Většinou je třeba do linuxového jádra implementovat základní hardwarovou funkcionalitu (výrobce zdrojové kódy systému neposkytuje). Naštěstí pro nás průzkumné kroky tohoto procesu udělali jiní, ale popíšeme si průběh celého postupu.

5.2.1 Seagate Dockstar

Připojení počítače k zařízení pomocí sériového portu

K tomu, abychom mohli nainstalovat do zařízení nějaký jiný firmware, je potřeba velice detailně prozkoumat jeho architekturu. Z vnějšího shlédnutí zařízení již víme, že obsahuje síťové rozhraní, několik portů USB, resetovací tlačítko a diodu LED. Z původního webového rozhraní zařízení dále lze vyčíst, že obsahuje procesor Marvell Kirkwood, 128 MiB RAM a 256 MiB Flash.

Tyto informace jsou bohužel naprosto nepostačující a proto nám nezbývá, než se podívat dovnitř zařízení. Zjišťujeme, že je na desce vyveden desetipinový konektor (s ne úplně standartní roztečí pinů 2.00 mm namísto 2.54 mm). Zkušenější vývojář správně odhadne, že zde budou vyvedeny JTAG signály, které jsou potřeba minimálně pro otestování funkčnosti desky a nahrání firmwaru na továrním páse. Bohužel nám však výrobce popis signálů jednotlivých pinů konektoru neposkytne, takže musíme například pomocí měřícího přístoje zjistit, jaký pin konektoru vede k jakému pinu procesoru a podle datasheetu k procesoru vyčíst speciální funkci pinu. Pokud se nám význam pinů nepovede zjistit, pak většinou pokus o nahrání jiného systému končí.

Zjistili jsme tedy, na kterých pinech jsou vyvedeny ladící signály JTAG rozhraní, dokonce jsme zjistili, že je zde vyveden sériový port. Propojíme tedy sériový port s počítačem pomocí vhodného převodníku (pamatujme, že jsou piny konektoru k procesoru připojeny přímo a I/O piny jsou uspůsobené pro napěťové signály 0 - 3.3 V).

Seznámení se systémem

Propojili jsme zařízení s počítačem přes sériové rozhraní a po připojení napájení k zařízení jsme vypozorovali, že jako první se zavede bootloader U-Boot. Nechali jsme nabootovat systém. Linuxové jádro vypisuje události na seriový port, který posléze převezme přihlašovací skript a my se můžeme nalogovat do systému jako uživatel root. Na zařízení je nahraná speciální distribuce linuxu pro NAS, obsahuje ale klasické nástroje, s jakými pracujeme v linuxu na desktopovém prostředí. Navíc obsahuje nástroje pro práci s pamětí Flash. Toho

využijeme a zazálohujeme si originální obsah paměti, kdybychom se později rozhodli nahrát tovární firmware zpět.

Podotkněme, že paměť Flash bývá virtuálně rozdělena na několik oddílů podobně, jako můžeme rozdělovat pevný disk. Je zde ovšem jeden podstatný rozdíl - rozdělení provádí většinou linuxové jádro pomocí speciální tabulky popsané ve zdrojovém kódu jádra[1]. To znamená, že nelze za běhu systému rozdělovat Flash, dokonce je nutné kvůli tomu znovu překompilovat jádro. V tabulce 5.2.1 je originální rozdělení paměti.

Bootloader U-Boot

Je velice příjemné zjištění, že zařízení bootuje právě pomocí tohoto bootloaderu, protože ve standartní konfiguraci poskytuje nespočet možností práce se zařízením. Pro nás je největší výhoda bootování systému linux přes síť. Na počítači připojeném přes síťové rozhraní k Dockstaru nainstalujeme DHCP (ten není nezbytný) a TFTP server a nastavíme tak, aby správně odpovídaly na požadavky. Dále pro testování prostředí je vhodné mít kořenový souborový systém na NFS serveru, takže opět nainstalujeme příslušný balík a nastavíme exportování vybraného adresáře.

Budeme potřebovat vybuildovaný systém - ten se skládá z linuxového jádra a kořenového souborového systému. Nejprve získáme zdrojové kódy projektu OpenWRT z repozitáře SVN, nakonfigurujeme pro Dockstar a spustíme překlad celého systému. Protože se kompilují překladové nástroje a následně celý systém, může překlad trvat delší dobu. Po skončení kompilace zkopírujeme soubor s linuxovým jádrem do adresáře, se kterým pracuje TFTP server. Také soubory kořenového souborového systému zkopírujeme do exportovaného NFS adresáře.

Nyní jsme připraveni nabootovat systém ze sítě, takže resetujeme zařízení a přerušíme automatické bootování systému stisknutím nějaké klávesy v terminálu napojeném na sériový port. Dostali jsme se do konzole U-Bootu. Pomocí příkazu `dhcp` získáme automaticky IP adresu pro zařízení a adresu TFTP serveru. Dále příkazem `tftp` stáhneme linuxové jádro ze síťě do paměti RAM. Nyní přepíšeme příkazovou řádku s nastavením pro linuxové jádro například takto:
`bootcmd="rootfstype=nfs nfsroot=192.168.0.1:/korenovyAdresar"`.
Nakonec stačí nabootovat linuxové jádro z přednastavené adresy v paměti příkazem `bootm` a pokud jsme správně nakonfigurovali jádro (především zakompilovali moduly pro síť), měli bychom za několik sekund získat výzvu na přihlášení do konzole.

[1]Konkrétně se provádí v souboru popisujícím určité zařízení. Je to soubor, kde se popisuje připojení všech periferií, například LED připojené přes GPIO, Flash paměti apod. které nejsou připojeny na sběrnici s dynamickou konfigurací (PCI aj.).

Název oddílu	Počátek	Konec	Velikost	Popis
U-Boot	0x00000000	0x000FFFFF	1 MiB	Bootloader
uImage	0x00100000	0x005FFFFF	4 MiB	Jádro Linux
root	0x00500000	0x024FFFFF	32 MiB	Kořenový systém souborů
data	0x02500000	0x0FFFFFFF	219 MiB	Místo pro uživatelská data

Tabulka 5.1: Rozdělení paměti Flash na oddíly

Shrňme si příkazy, které jsme provedli, pro nabootování systému ze sítě:

1. `dhcp` - Získání všech potřebných síťových adres a informací.
   Správným nastavením DHCP serveru lze počet nutných manuálních kroků při procesu bootování ze sítě omezit na tři. Lze nakonfigurovat adresu TFTP serveru, potom nemusíme pro U-Boot specifikovat IP adresu serveru. Dále lze pro konkrétní MAC adresu předurčit název souboru s linuxovým jádrem, opět tím odpadá nutnost v příkazu `tftp` specifikovat soubor ke stažení.

2. `tftp` - Stažení souboru ze sítě do paměti RAM. Pokud nespecifikujeme adresu v paměti, soubor se nahraje na standartní umístění (místo, kam by nahrával bootloader obraz linuxového jádra při automatickém bootování).

3. nastavení proměnné `bootcmd` - je nutné nastavit zařízení obsahující kořenový souborový systém a typ souborového systému. Je také vhodné nastavit výpis do terminálu, abychom mohli zjistit případný problém při bootování.

4. `bootm` - spuštění obrazu jádra v paměti.

Zapsání systému OpenWRT na Flash

Máme funkční jádro a souborový systém, takže už máme většinu práce za sebou. Zbývá nám nahrát obraz jádra a souborový systém do paměti Flash. Následující kroky jsou nebezpečné a proto je nutné dbát na jejich přesné provedení, jinak je možné způsobit poškození obsahu paměti. Pokud způsobíme přehrání bootloaderu, po restartu zařízení již nebude fuknční a systém nenabootuje. Z tohoto stavu je sice možné se dostat zpět, ale je nutné použít rozhraní JTAG a pomocí něho buď zapsat kód zavaděče do paměti a spustit jej, nebo přímo zapsat zavaděč na Flash.

Opět přerušíme standartní bootovací proces zařízení a dostaneme se do konzole U-Bootu. Získáme IP adresu pomocí `dhcp`. Prostor `0x01000000 - 0x01100000` v paměti RAM budeme zapisovat do Flash, takže si jej pro jistotu nejprve vymažeme příkazem

```
mw 0x01000000 0 0x00100000
```

Přeneseme soubor s linuxovým jádrem, pomocí příkazu

```
tftp 0x01000000 vmlinuz
```

Dále vymažeme 4 MiB ve Flash:

```
nand erase 0x00100000 0x400000
```

a zapíšeme stažený soubor z RAM do flash pomocí

```
nand write.e 0x800000 0x100000 0x100000
```

Podobný postup provedeme i pro oddíl s kořenovým systémem sobourů, stáhneme přes TFTP soubor root-jffs2-128k (Ten obsahuje systém souborů JFFS2 určený pro Flash, který je zarovnaný na 128 KiB). Takže v terminálu provedeme příkazy:

```
tftpboot 0x1000000 root-jffs2-128k

nand erase 0x500000 0x2000000

nand write.e 0x1000000 0x500000 0x200000
```

Je nutné dbát na to, aby nebyla přepsána část paměti Flash s bootloaderem U-Boot! Proto je jistější dvakrát zkontrolovat hodnoty, které píšeme do konzole a porovnat je s rozdělením paměti Flash podle tabulky.

5.2.2 Instalace systému OpenWRT na Avila

Přestože v rámci práce bylo nainstalovat systém OpenWRT pouze na jedno zařízení, rozhodli jsme se o rozšíření práce a zprovoznili jsme systém OpenWRT sestavený námi i na druhém zařízení. Jak se následně ukáže, postup nainstalování OpenWRT byl u tohoto zařízení o něco jednodušší než u Seagate Dockstar, třeba i proto, že Avila Board je vývojová platforma a ne hotový produkt.

Umístění nového systému na kartu

Protože tato deska obsahuje slot na kartu typu Compact Flash, pokusíme se zachovat originální systém nahraný v paměti Flash na desce a nový systém spustit z karty. Tento postup má velkou výhodu v tom, že zařízení nemusíme nechat bootovat ze sítě, abychom vyzkoušeli funkční obraz před jeho zápisem, protože stále budeme mít fuknční systém.

Propojení pomocí sériového portu

K propojení s počítačem pomocí sériového portu stačilo použít sériový kabel. Na desce je totiž vyvedený konektor s rozhraním RS232 a tím pádem obsahuje i převodník logických úrovní. Po připojení napájení jsme zjistili, že o start zařízení se stará bootloader RedBoot, který je postaven na real-time operačním systému eCos (embedded configurable operating system). Na první pohled je, co se funkcionality týká, více vybavenější než bootloader U-Boot (například umožňuje jednoduše pracovat s více obrazy systému, či jednoduchou manipulaci s PCI zařízeními), ale kvůli tomu také jeho konfigurace náročnější.

Nahrání a nabootování nového systému

Po nabootování systému vytvoříme na detekované CF kartě dva oddíly nástrojem fdisk a vytvoříme na nich souborový systém. Na první nahrajeme soubor s linuxovým jádrem získaný při sestavení obrazu OpenWRT. Na druhý rozbalíme archiv s kořenovým souborovým systémem.

Abychom tento náš nový systém zavedli, při restartování desky přerušíme automatické bootování a provedeme tyto příkazy:

```
load -r -m disk -b 0x01600000 hda1:zImage
exec -c "console=ttyS0,115200 root=/dev/sda2"0x01600000
```

První příkaz nahraje soubor zImage s linuxovým jádrem z prvního oddílu CF karty do paměti RAM. Druhý spustí kód jádra s parametry v uvozovkách z příslušné adresy RAM.

Kapitola 6

Rozšíření pro síťovou bezpečnost

Jako rozšíření byla vybrána implementace algoritmu vyhledávající vzory v paketech. Konkrétně se jedná o algoritmus Delay DFA, popsaný v kapitole 4.4.1.

6.1 Vytvoření jednoduchého rozšíření pro OpenWRT

Jako jednoduché rošíření v rámci semestrálního projektu byl implementován program, který v paketech vyhledává konkrétní vzor. K vyhledávání bylo využito velice jednoduchého alogritmu, jehož zdrojový kód v jazyce C vypadá následovně:

```
int match_pattern(const u_char *packet, int len)
{
    int i;
    while(len > pattern_len)
    {
        for(i = 0; i < pattern_len; i++)
        {
            if(packet[i] != pattern[i])
                break;
            if(i == pattern_len - 1)
                return 1;
        }
        packet++;
        len--;
    }
    return 0;
}
```

Zde nebylo nutné testovat výkonnost algoritmu, neboť tato naivní implementace sloužila pouze k vyzkoušení základních principů pro programování na vestavěném zařízení. Vyzkoušeli jsme si tak, co je potřeba při kompilaci a sestavení aplikací určenou pro jinou platformu.

6.1.1 Knihovna libpcap

Aby program mohl získávat pakety procházející jádrem, bylo potřeba použít knihovnu zachytávající tyto pakety. Knihovna *libpcap* je nejznámější implementací pro tento problém.

Vlastní získávání paketů pomocí knihovny je rozděleno do několika kroků:

1. Vybereme si rozhraní, ze kterého chceme pakety získávát. V naší konkrétní implementaci jsme nechali knihovnu, aby vybrala za nás standartní rozhraní, pomocí funkce `pcap_lookupdev`. V systému linux je to typicky rozhraní `eth0`.
2. Získáme objekt rozhraní, se kterým budeme manipulovat (`pcap_open_live`).
3. Vstoupíme do nekonečného cyklu funkce `pcap_loop`. Ta sama zachycuje pakety a předává je uživatelské funkci.

Naše uživatelská funkce přijímá získané pakety (za každý získaný paket vypíše tečku), následně volá funkci `match_pattern` a při pozitivním výsledku (nálezu daného podřetězce) vypíše základní informace o paketu.

6.1.2 Integrace do OpenWRT

Abychom mohli využít automatizovaného prostředí OpenWRT pro kompilaci nástroje pro konkrétní architekturu, bylo potřeba vytvořit několik souborů pro automatickou kompilaci, pomocí balíku nástrojů *Autotools*. Je nutné ve speciálním souboru specifikovat, jaké balíky a knihovny jsou potřebné pro náš program (pro nás je to knihovna *libpcap*). Ve druhém nezbytném souboru jsou jména souborů a definice proměnných, které má použít kompilátor. Po vytvoření obou těchto souborů již nic nebrání vygenerování skriptu *configure* za pomocí třech nástrojů `aclocal`, `automake`, `autoconf`.

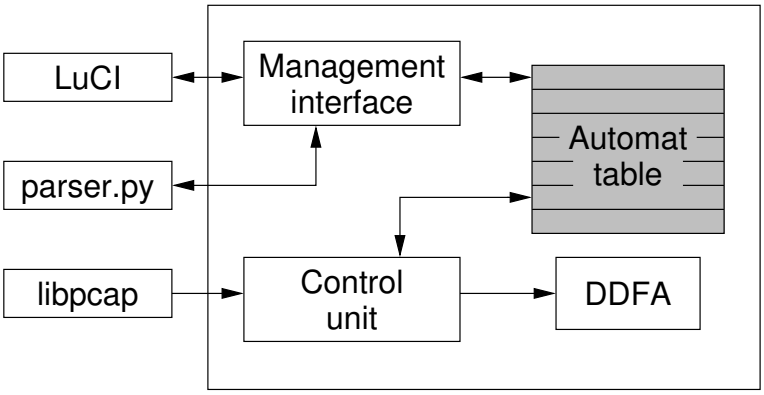
Takto vytvořený balíček je nyní nutno vložit do překladového systému OpenWRT. V adresáři *packages* vytvoříme novou složku s názvem balíčku a v ní soubor Makefile. Popíšeme náš balíček pomocí několika proměnných (především název balíčku, verzi, adresu ke stažení, kontrolní součet MD5 a základní popis). Nakonec spustíme konfiguraci systému a tento nově přidaný balíček vybereme, takže nyní se po spuštění sestavení celého systému náš balík zkompiluje pro danou architekturu a začlení do kořenového adresáře.

6.2 Návrh rozšíření pro síťovou bezpečnost

Ačkoliv se nejedná o rozsáhlé rozšíření, i malý modul je třeba správně navrhnout. Bylo nutno si ujasnit, co vše od rozšíření čekáme:

- Modul získává pakety protékající systémem.
- Na těchto paketech provádí prohledávání daných vzorů.
- Je dostupná statistika počtu shod pro každý vzor.
- Je možné přidávat a odebírat vzory dynamicky, při prohledávání provozu.

Do procesu prohledávání vzorů zahrneme všechny pakety procházející linuxovým jádrem, bez ohledu na vyřazení paketu v ostatních modulech systému. Za tohoto předpokladu můžeme použít již zmíněnou knihovnu libpcap.



Obrázek 6.1: Schéma modulu rozšíření

Jádrem rozšíření bude modul přijímající data, která se mají prohledat a množina konečných automatů, každý z nich testuje dané regulární výrazy.

Konečný automat se generuje z regulárního výrazu pomocí programu napsaném v jazyku Python. Převedením vznikne textový řetězec složený z několika částí popisujících a) abecedu, b) přechody, c) stavy a d) počáteční stav a koncové stavy automatu. Bohužel pro dosažení co nejoptimálnějšího konečného automatu analýza a převod regulárního výrazu trvá poměrně dlouhou dobu – pro jednoduché výrazy trvá řádově v jednotkách, pro složité výrazy řádově v desítkách sekund.

Při implementaci použijeme vícevláknové zpracování. Prvním důvodem je možnost přidávat vzory dynamicky za běhu rozšíření a to by s jedním vláknem nebylo možné právě kvůli pomalému generování automatu. Druhým důvodem je využití případných vícejádrových procesorů, na každý by tak mohla připadnout určitá část množiny automatů. Tento počet vytvořených vláken bude nastavitelný.

Jednotlivá vlákna tedy budou obsluhovat tyto části:

1. Obsluha rozhraní LuCI – zpracování vstupních příkazů a předávání statistik.
2. Generování konečného automatu pomocí skriptu v jazyce Python.
3. Rozdělování jednoho paketu mezi jednotlivé automaty.
4. Testování všech sledovaných vzorů na paketu – simulace na všech automatech.

Výsledné schéma modulu je znázorněno na obrázku 6.1.

Co se týká optimalizací, nebylo zde využito žádných hardwarových jednotek a to ze dvou důvodů. Prvním je, aby program mohl bežet na libovolné platformě. Tím druhým je ten, že na platformě ARM Kirkwood byly pouze akcelerační jednotky pro výpočet CRC a šifrování, žádný z těchto bloků nelze rozumě v algoritmu Delay DFA použít.

Kapitola 7

Testování výkonnosti směrování

7.1 Metodika měření

Pro měření výkonu routerů jsme použili jednoduchý linuxový nástroj iperf, který dokáže měřit propustnost sítě pomocí protokolu TCP nebo UDP. Je zapotřebí mít v testované síti dva počítače, na jednom z nich se spustí iperf v režimu serveru, na druhém se iperf sputi jako klient, navíc se mu předá IP adresa serveru. Klient se k tomuto serveru připojí a snaží se serveru odesílat pakety, jak nejrychleji systém dovolí. Po nastavené době přestane odesílat měřící pakety a se serverem si vymění statitiky, které následně oba vypíšou.

V tomto testu byla použita standartní doba, což je 10 s. Měřili jsme propustnost pro paketové délky 64, 256, 512, 1024 a 1500. Pro reálnější výsledky jsme navíc každé měření zopakovali 5x a hodnoty zprůměrovali. V příloze B jsou kompletní tabulky měření, zde jsou pouze průměrné hodnoty vyneseny do grafu.

7.2 Postup měření

Nejprve jsme si ověřili rychlost samotné linky mezi dvěma počítači, bez jakéhokoliv dalšího zařízení v cestě.

Poté jsme testovali rychlost switchů, které obsahují pouze routery Linksys a D-Link. Pomocí nich propojují rozhraní LAN, data tudíž vůbec neprojdou do hlavního procesoru, ale jen do speciálního obvodu, který provádí vlastní switchování. Předpokládáme, že tento obvod by měl způsobit jen nepatrné zpoždení a tím pádem bude propustnost při použití TCP protokolu ovlivněna jen mírně (díky velikosti TCP okénka a round-trip zpoždění). Při použití UDP by propustnost neměla být ovlivněna vůbec.

Dále jsme měřili rychlost routování. Klasické routery jsme otestovali s vypnutým i zapnutým firewallem. U vývojové desky Avila, která je standartně dodávaná se systémem OpenWRT, bylo k routování použito nástrojů iptables. Tyto nástroje zároveň fungují i jako firewall, který v podstatě vypnout nelze. U iptables jsme nastavili pouze pravidla pro přeposílání paketů z jednoho rozhraní na druhé a naopak.

K síťovému disku Dockstar jsme museli přistupovat jiným způsobem, protože obsahuje jen jediné Ethernetové rozhraní. Proto jsme na něm zprovoznili systém OpenWRT a nástroj iperf. Propojili jsme ho napřímo s počítačem a postupně jsme na síťovém disku spustili nástroj iperf, nejprve jako server a pak jako klient. Tak jsme ověřili, jak rychle dokáže data vysílat i přijímat.

7.3 Měření pomocí protokolu TCP

Měření propustnosti pomocí protokolu TCP bylo jednoduché a přímočaré. Z naměřených hodnot a grafů 7.5 je vidět 7.5, že náš předpoklad ohledně zpoždění switche se potvrdil.

Dále můžeme vyčíst, že nejúspěšnější byl v tomto testu Dockstar s nejrychlejším procesorem. Je však ale také důležité připomenout, že Dockstar neprováděl routování, ale jen přijímání nebo odesílání paketů. Pokud bychom tedy uvažovali routování a následné odesílání paketů, výkon by značně klesl. Na druhou stranu procesor v tomto zařízení narozdíl od ostatních není síťový, architektura jeho jádra tedy není optimalizovaná a nemá žádné akcelerační jednotky pro zpracování síťového provozu.

Poslední informací, kterou můžeme vidět z grafu je, že pouhé povolení firewallu (bez žádného pravidla) znamenalo jistou výkonovou ztrátu v jednotkách procent. Lze si domyslet, že přídáním několika pravidel by výkon dále klesal.

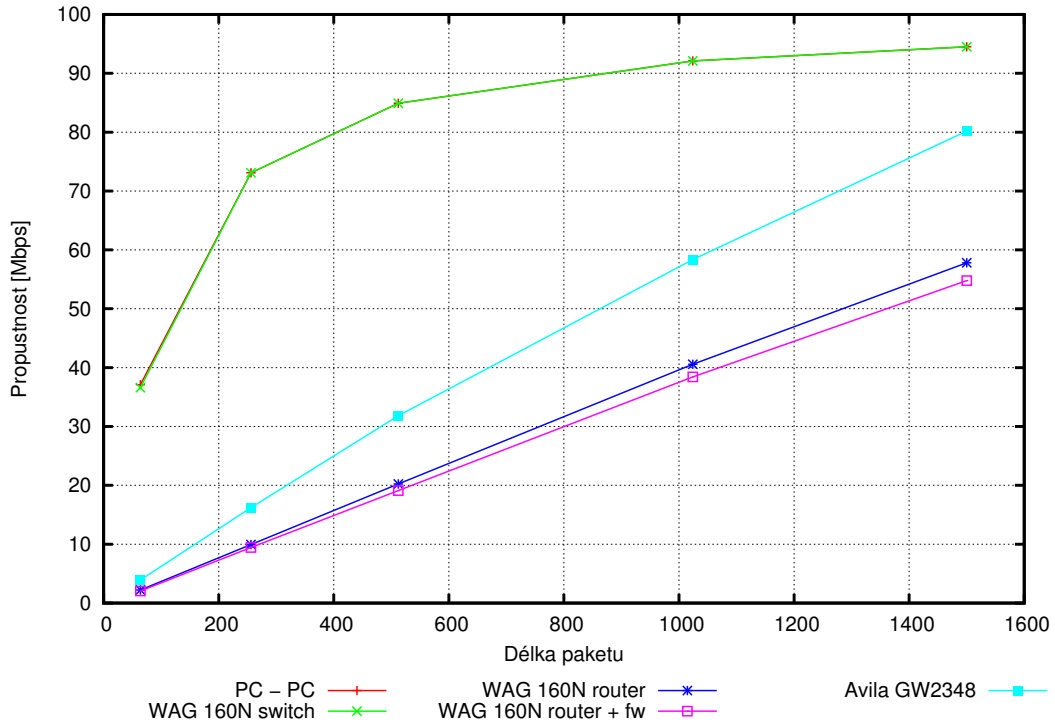
7.4 Měření pomocí protokolu UDP

Měření pomocí protokolu UDP bylo o dost více komplikované než u TCP. Protože odchylky jednotlivých měření jsou více než únosné, tyto hodnoty nemají žádnou přesnou vypovídající hodnotu a je důležité se k nim tak také stavět.

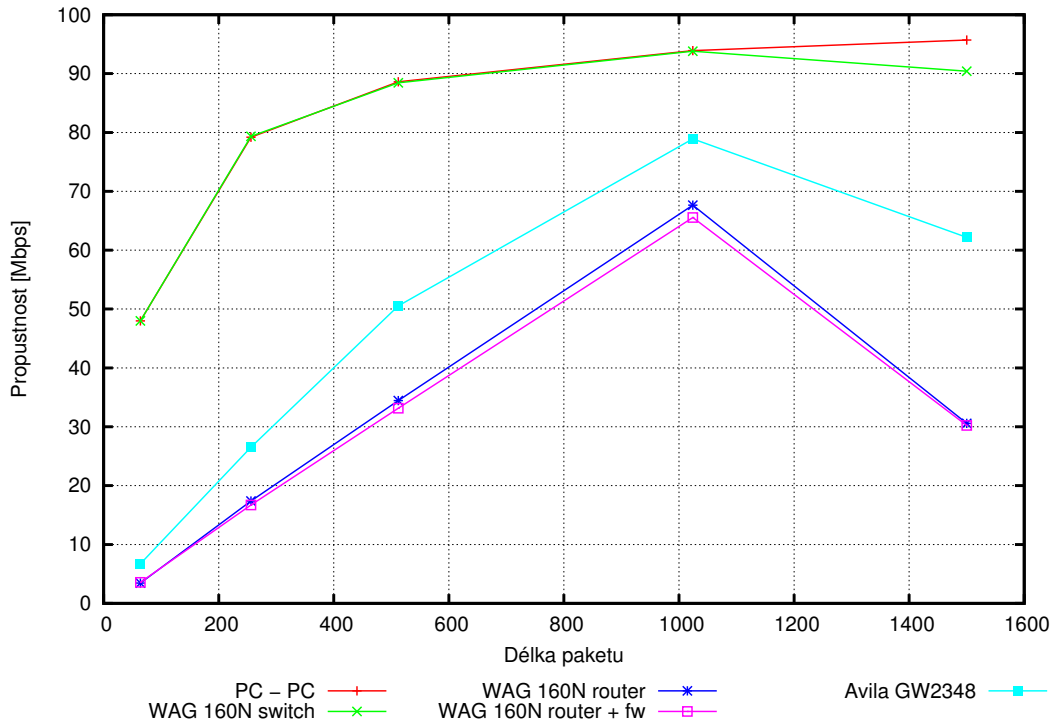
Je vidět, že funkce switche nyní neovlivnila hodnoty měření UDP dle přepokladu alespoň pro nižší rychlosti, kde chyby jednotlivých měření byly v rozumných mezích.

7.5 Závěr měření propustnosti při směrování

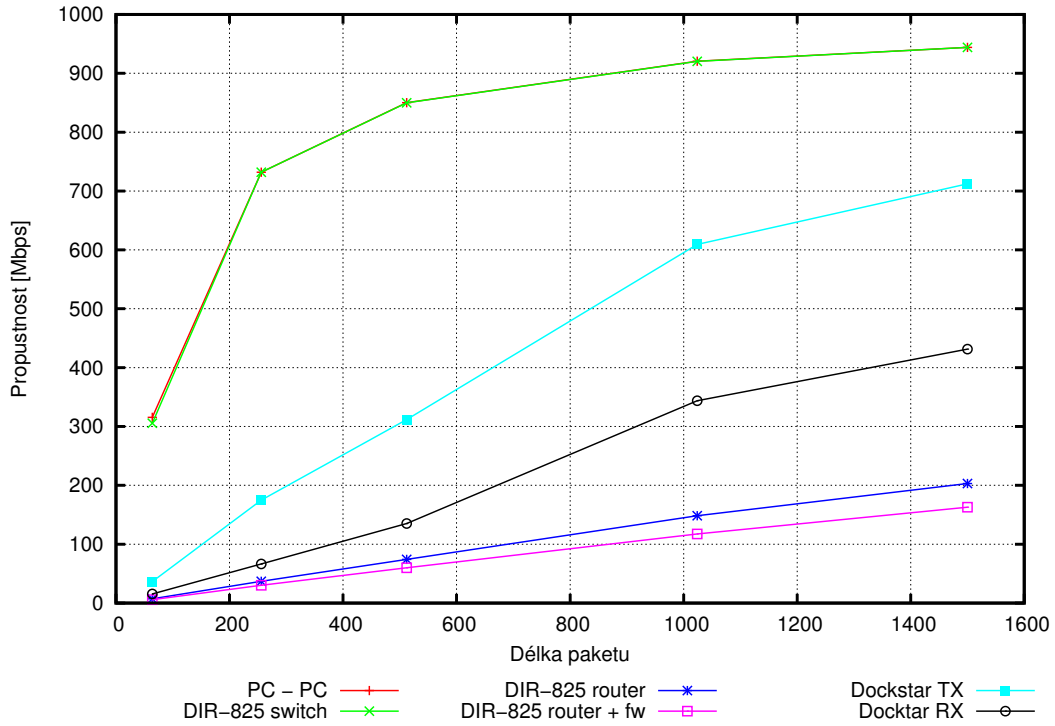
Závěrem měření byl fakt, že tyto domácí routery nejsou výkonné natolik, aby zpracovaly síťový provoz v takové rychlosti, kterou poskytuje jejich Ethernetové rozhraní. Je nutné si uvědomit, že někteří postkytovatelé ve velkých městech již nabízejí možnost internetové konektivity pro domácnosti s rychlostí 100 Mb a tedy že tento přístup domácích routerů je nutné co nejdříve změnit.



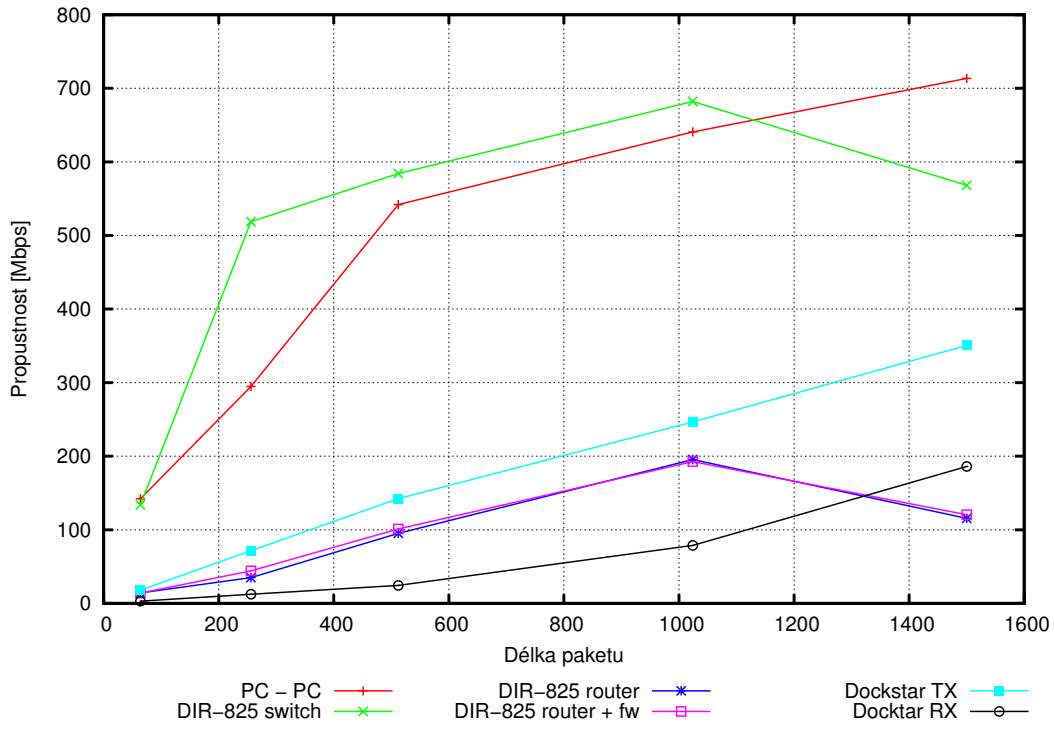
Obrázek 7.1: Graf propustnosti pro zařízení s 100 Mb Ethernetem, protokol TCP



Obrázek 7.2: Graf propustnosti pro zařízení s 100 Mb Ethernetem, protokol UDP



Obrázek 7.3: Graf propustnosti pro zařízení s 1 Gb Ethernetem, protokol TCP



Obrázek 7.4: Graf propustnosti pro zařízení s 1 Gb Ethernetem, protokol UDP

Kapitola 8

Testování výkonnosti na algoritmech

Pro účely této práce byly v jazyce C implementovány algoritmy popsané v kapitole 4. Každý algoritmus pak byl doplněn o modul, jehož účelem bylo předat algoritmu vstupní data, spustit jej a změřit čas provádění algoritmu, popř. jeho částí.

Tyto programy byly přeloženy pro testované procesory (pomocí tzv. toolchainu dostupného v OpenWRT) a spuštěné na konkrétních zařízeních. Pro překlad byly použity standardní hodnoty překladového systému, což v případě OpenWRT znamená, že spustitelné soubory jsou optimalizovány tak, aby měli co nejmenší velikost[1]. Přestože výkonnostní dopad optimalizací je veliký (což si také na konci kapitoli předvedeme), standardní hodnoty pro testy na všech zařízeních jsme neměnili a ponechali optimalizaci na velikost. Hlavním důvodem byla skutečnost, že pro jedno zařízení jsme měli k dispozici hodnoty naměřené s testy algoritmů optimalizovaných na velikost avšak neměli jsme možnost otestovat tyto algoritmy optimalizací na rychlost.

Abychom předešli velké latenci při čtení z dat z disku, sítě nebo Flash paměti (volba záleží na konkrétním zařízení a běhu OpenWRT), načetli jsme si nejprve celý soubor do paměti a až poté spustili test. Jinak bychom dostali při měření irelevantní hodnoty.

V grafech následujících kapitol je vyneseno výkonnostní hodnocení takovým způbosem, že větší hodnota znamená vyšší výkonnost. Většinou jsou hodnoty vypočítány podle vzorce

$$y = k\frac{s}{ft}$$

kde s je velikost dat, f je frekvence procesoru a t je naměřená doba běhu části algoritmu, nebo některé jeho části. Tímto způsobem je možné z grafu přímo porovnávat architektury procesorů mezi sebou, nehledě na frekvenci (s určitým přihlednutím k subsystémům, které výsledky měření mohly ovlivnit, především paměť RAM a její časování).

V grafech v následujících kapitol se také vyskytuje zařízení s názvem Esprimo. To je osobní počítač s procesorem Intel Core2Duo taktovaný na frekvenci 2,4 GHz.

[1] Překladač gcc pro optimalizaci na velikost používá parametr -Os

8.1 Algoritmy s Bloomovy filtry

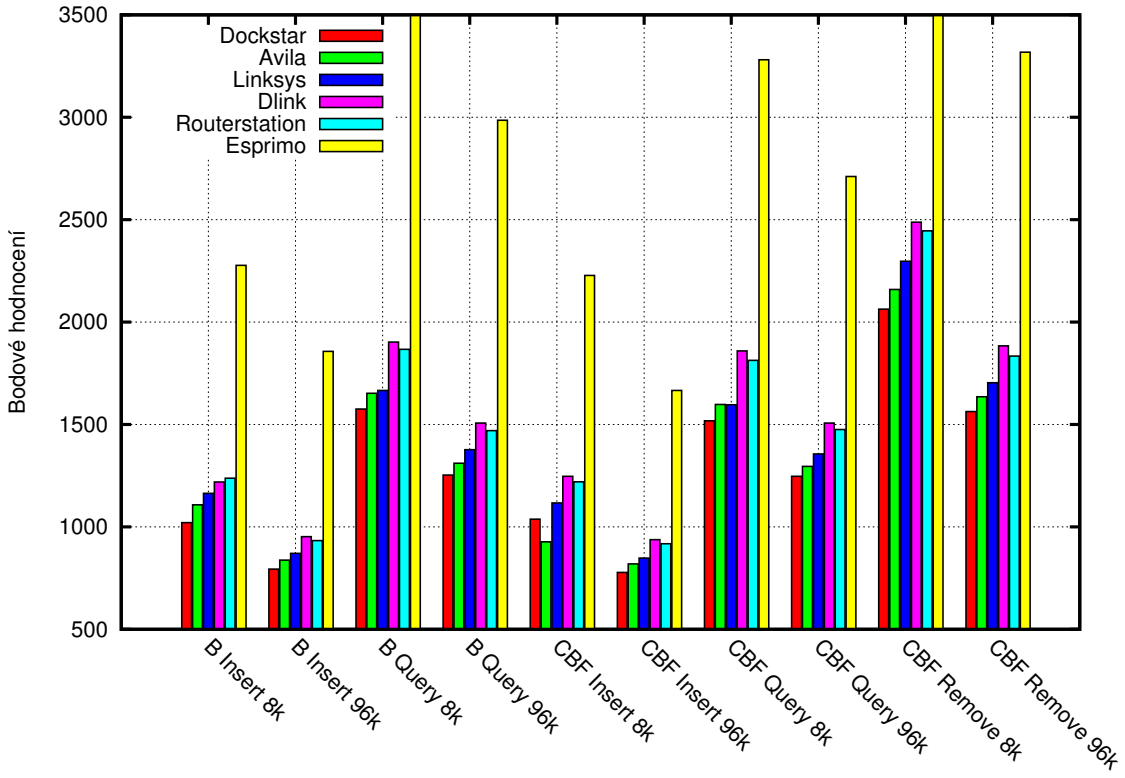
Prvním testem je základní implementace Bloomových filtrů. Test je rozdělen na dvě části: Část plnění, kdy je každá datová položka vložena do bitového pole Bloomova filtru. Druhá část je dotazování na položky, které byly náhodně zvolené z vloženého seznamu, těch byla polovina z počtu vložených položek.

Druhým testem je implementace čítajících Bloomových filtrů. První dvě části algoritmu jsou z hlediska testu shodné, přidáváme ovšem další dvě části. Třetí část je tedy odstranění položky, čtvrtou částí je opět dotaz na náhodně zvolné položky, tentokrát ze seznamu odstraněných položek.

Přestože byly měřeny čtyři datové sady, do grafu 8.1 byly zahrnuty pouze dvě pro každou část algoritmů. Odůvodněno je to tím, že mezi páry datových sad (8 kB – 32 kB a 96 kB – 384 kB) byly téměř nepozorovatelné změny a graf by s nimi pouze ztratil na přehlednosti.

Povšimněme si však, že se datové sady o velikosti 8 kB a 96 kB liší v naměřených výsledcích asi s 20% ztrátou výkonnosti. U datových sad větších než 32 kB se totiž začínají projevovat výpadky cache.

Je zde také vidět, že zařízení Routerstation má o něco menší hodnocení než zařízení D-Link. Oba sice mají shodný procesor, ale v zařízení Routerstation je přetaktován na frekvenci 720 MHz. Za snížené hodnocení zde může taktovací frekvence pamětí, která je u obou zařízení shodná. Výkonově je přetaktovaný procesor samožřejmě výhodnější – test na tomto zařízení proběhl rychleji.



Obrázek 8.1: Graf hodnocení výkonu v algoritmech s Bloomovými filtry

8.2 Algoritmy provádějící LPM

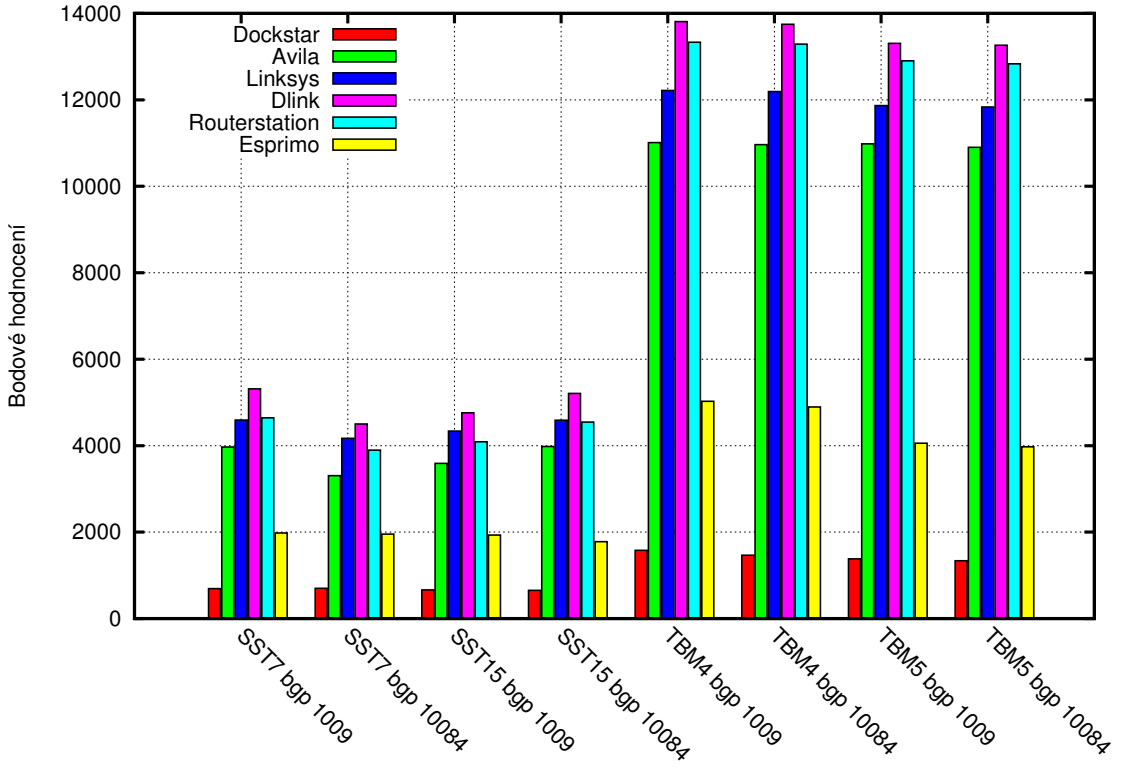
Jako základ pro testy algoritmů SST a TBM posloužily dvě BGP tabulky: jedna obsahovala 1009 položek, druhá 10084 položek. Tabulky byly vždy převedeny do formy, kterou přijímal konkrétní algoritmus.

Oba algoritmy je možné parametrizovat jednou konstantou. U SST se jedná o maximální aritu jednoho uzlu, byly použity hodnoty 4 a 7. U TBM je to výška podstromu v původním grafu, ze kterého se buduje jeden uzel, použili jsme hodnoty 4 a 5.

Opět byly testovány čtyři datové sady s rostoucí velikostí. Nyní se však již díky charakteristice algoritmu neprojevovaly výpadky cache na výkonnostním hodnocení, takže byly do grafu zahrnuty pouze výsledky týkající se jediné (nejmenší) datové sady. Na grafu 8.1 je vidět opět několik zajímavostí.

Na první pohled je jasně vidět, že algoritmus TBM je téměř třikrát výkonnější, než algoritmus SST.

Dále si můžeme všimnout, že v tomto testu různá architektura procesorů silně ovlivnila výsledky. Procesory založené na jádru MIPS několikrát výkonnostně převyšují architekturu Intel x86 a ARM. Naproti tomu procesor ARM Kirkwood s jádrem Sheeva nepropadá vůči procesoru x86 tolikrát, jako v testu Bloomových filtrů. Co se týká výkonnosti procesoru Intel IXP425, v tomto testu překvapil svým hodnocením, protože výkonnostně dosahuje výsledků procesorů MIPS. Jedná se sice o síťový procesor a tak by měla být jeho architektura optimalizovaná i například pro algoritmy podobné LPM, ale tento procesor má instrukční sadu ARM5vTE, kompletně shodnou s ARM Kirkwood, který výkonnostně dopadl jasně nejhůře.



Obrázek 8.2: Graf hodnocení výkonu v algoritmech s LPM

Jako poslední si můžeme všimnout, že velikost směrovací tabulky ovlivňuje dobu vyhledání nejdelšího prefixu minimálně. Algoritmus TBM dosahuje větší výkonnosti pro výšku podstromu 4 nezávisle na velikosti BGP tabulky, u algoritmu SST takto univerzální nejlepší parametr určit nelze.

8.3 Algoritmy pro vyhledávání vzorů

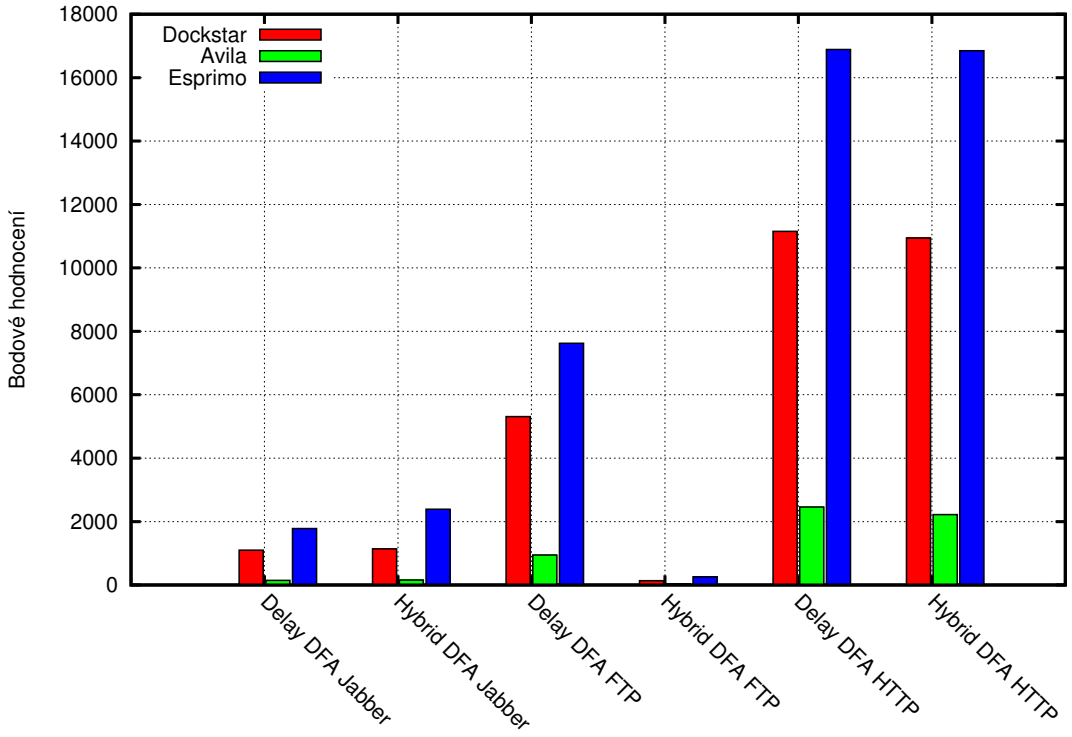
Byly zde testovány algoritmy Delay DFA a Hybrid DFA. Testovaná data jsou společná pro všechny testy a obsahují asi 1 Mb síťového provozu zahrnující několik druhů komunikací. Alespoň jednou se v testovacích datech objevil každý vyhledávaný regulární výraz.

Pro generování automatů byly využity následující regulární výrazy:

- `[GE|PO]ST` – vyhledává pakety protokolu HTTP
- `/<stream:stream [ -~]*xmlns=['"]jabber` – vyhledává pakety protokolu Jabber
- `/^CWD\s[^\n]*?\.\.\./smi` – jeden ze sedmi výrazů pro rozpoznání protokol FTP

Zde jsme měli možnost otestovat pouze dvě zařízení s procesorem ARM a osobní počítač s procesorem x86.

V tomto testu je opět výkonově nejlepší architektura Intel x86. Avšak zajímavé je, že v tomto testu zařízení Seagate Dockstar dopadlo několikrát lépe než platforma Avila. V předchozích testech přitom výsledky těchto dvou zařízení byly úplně opačné. Lze tedy říci, že jádro Sheeva určené pro všeobecné použití je pro testování vzorů pomocí konečných automatů výhodnější než jádro XScale, které je specializované na sítě.



Obrázek 8.3: Graf hodnocení výkonu v algoritmech s vyhledáváním vzorů

Z výkonnostního hlediska algoritmů nejlépe dopadl algoritmus Delay DFA. Proto také byl vybrán pro implementaci rozšíření.

Algoritmy se výrazně liší výsledky pouze u vyhledávání vzorů protokolu FTP. Je snadné nahlédnout, že algoritmus Hybrid DFA má problémy s rozvětvenými automaty. Také konfigurační soubor popisující tento automat byl několikrát větší, než pro druhý algoritmus.

8.4 Vliv optimalizací

Jak již bylo řečeno, hodnoty vynesené v grafech předchozích podkapitol byly naměřené pro programy optimalizované na velikost. Chceme však ukázat, kolik výkonu OpenWRT ztrácí, když při sestavování systému používá optimalizaci na velikost.

Následující tabulka tedy obsahuje:

a) Hodnoty zrychlení, které jsou podílem zprůměrovaných časů běhů optimalizovaných a neoptimalizovaných programů[2].

b) Poměry velikostí výsledného spustitelného souboru (bez vstupních dat) udávají, kolikrát je větší program optimalizovaný na rychlost, než program optimalizovaný na velikost. Výsledné hodnoty velikostí pro architekturu ARM lze sloučit, neboť procesory mají stejnou instrukční sadu (zde je rozdíl pouze v uspořádání bytů ve slově, tzv. Endianness) a tedy měly by mít i shodnou velikost. Toto tvrzení se potvrdilo i prakticky.

Algoritmus	Zrychlení			Poměr velikosti kódu	
	Dockstar	Avila	Esprimo	ARM	Intel x86
Bloom Filter	3.57	3.77	1.02	1.02	1.02
Counting Bloom Filter	3.53	3.68	1.02	1.03	1.02
Shape Shifting Trie	3.03	2.56	1.30	1.01	1.02
Tree Bitmap	3.14	2.97	1.14	1.01	1.02
Delay DFA	1.23	3.41	1.08	1.06	1.03
Hybrid FA	1.45	3.33	1.68	1.05	1.03

Tabulka 8.1: Zrychlení běhu programu a zvětšení souboru po změně optimalizací

Je vidět, že změna optimalizační metody pomohla hlavně platofmám s procesorem ARM, zajímavé také však je, že architekturu Intel x86 neovlivnily skoro vůbec. Nejméně se optimalizace dotka algoritmů pro vyhledávání vzorů u procesoru Kirkwood, jinak ostatní algoritmy přibližně třikrát výkonnější. Nárůst velikosti souborů byl přitom minimální, téměř neznatelný.

Z tohoto měření plyne, že nemá smysl provádět optimalizace programu na velikost, pokud přímo nenarazíme na případ, kdy se nám výsledný systém nevejde do vnitřní paměti zařízení. I v případě, že by se toto stalo, optimalizace pravděpodobně tento problém nevyřeší. Vývojáři systému OpenWRT by měli zvážit, zda nepřenastavit výchozí nastavení optimalizací pro kompilaci programů a sestavení celého systému.

[2]Měření optimalizovaného algoritmu u Bloomových fitrů probíhalo opět v několika krocích, jak tomu je v příslušné podkapitole. Ovšem v tabulce je uvedena pouze jedna hodnota, protože hodnoty zrychlení jednotlivých kroků se téměř shodovaly.

Kapitola 9

Další vývoj

V této kapitole si ukažeme, zda s výsledky, kterých jsme dosáhli, je možné nějakým způsobem pracovat na těchto platformách dále.

Viděli jsme, že optimalizace daly platformám nový výkonnostní rozhled. Lze bez obav tvrdit, že pro spoustu aplikací mají platformy větší potenciál než platforma x86. Otázkou je, zda-li bude možné na platformě ARM zvyšovat frekvenci podobným tempem, jako na platformě x86.

9.1 Výsledky algoritmu Delay DFA

Z výsledků testů algoritmu Delay DFA lze vidět, že pro vyhledávání vzorů v běžném síťovém naprosto nedostačuje. Už na testech samotného algoritmu zjišťujeme, že zařízení nebylo schopné zpracovat dostatečné množství dat, systém však navíc ještě musí řídit přijímání a odesílání paketů, což je, jak vyplývá z testů propustnosti, také netriviální úloha.

I přes optimalizace byl Seagate Dockstar schopen prohledat pouze 560 kB dat za 75 ms, což je přibližně 7.5 MB/s na nejjednodušším testovaném vzorku. Gateworks Avila dokonce jen jednu desetinu. Z toho plyne, že takovýto způsob prohledávání v reálném čase například 100 Mb provozu je nemyslitelný.

9.2 Hardwarová akcelerace

Naskýtá se zde možnost použít nějaký způsob hardwarové akcelerace. Existují například výkonné algoritmy pro vyhledávání vzorů implementovatelné v FPGA. Některé z nich jsou publikované s výbornými výsledky (jednotky Gbps): Bit-Split [5], B-FSM [7], Field-Merge [12].

Platforma Gateworks Avila obsahuje čtyři sloty pro karty MiniPCI, dnes není problém vyrobit kartu s FPGA komunikující přes sběrnici PCI. Síťový procesor by pak do této karty posílal jednak kompletní síťový provoz, jednak konfigurační data konečných automatů. Maximální teoretická přenosová kapacita této sběrnice je 133 MB/s (32 bit sběrnice taktovaná na 33 MHz), což by pro síť o rychlosti 100 Mbit plně dostačovalo.

Kapitola 10

Závěr

V této práci jsme ukázali, která zařízení najdeme v počítačových sítích, zabývali jsme se podrobněji síťovými routery a představili jsme si problémy, které řeší. Popsali jsme procesory s jádrem ARM a předvedli jsme, jaké konkrétní vlastnosti nabízí vybraná zařízení osazená těmito typy procesorů. Rozebrali jsme si algoritmy, které se používají v problematice síťové bezpečnosti.

V praktické části jsme na vybrané zařízení nainstalovali operační systém pro správu sítě OpenWRT. Přestože v rámci práce bylo nainstalovat systém OpenWRT pouze na jedno zařízení, rozhodli jsme se práci rozšířit a zprovoznili jsme námi sestavený systém i na druhém zařízení. Díky tomu bylo možné srovnávat výsledky dvou zařízení s procesorem ARM. Dále jsme do distribuce OpenWRT přidali modul, pro vyhledávání vzorů v síťových paketech.

Poslední kapitoly práce obsahují výsledky dosažené při měření. První z provedených měření byla výkonnost routerů při klasickém směrování paketů. Dále jsme oproti zadání otestovali nejen algoritmus Delay DFA ale i všechny další algoritmy uvedené v teoretické části. Poukázali jsme na některé zajímavé výsledky různých platforem.

Došli jsme k závěru, že domácí routery nejsou výkonné natolik, aby zpracovaly síťový provoz, který může generovat běžná domácí síť s internetovou konektivitou o rychlosti 100 Mb. Vzhledem k tomuto faktu nemá smysl, aby navíc zpracovávali provoz náročnějšími funkcemi a proto byla ukázána jedna z možností vylepšení, a to rozšíření platformy Avila o hardwarově akcelerovanou jednotku.

Literatura

[1] Barr, M.: *Programming Embedded Systems in C and C++*. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., 1999, ISBN 1-56592-354-5.

[2] Cisco Systems, I.: *CCNA 1 and 2 Companion Guide, Third Edition*. 201 West 103rd Street, Indianapolis, IN 46290 USA: Cisco Press, 2003, ISBN 1-58713-110-2.

[3] Eatherton, W.; Varghese, G.; Dittia, Z.: Tree bitmap: hardware/software IP lookups with incremental updates. *Computer Communication Review*, ročník 34, č. 2, 2004: s. 97–122.

[4] Furber, S. B.: *ARM System-on-Chip Architectue*. Reading, Massachusetts: Addison-Wesley Professional, 2001, ISBN 0-201-67519-6.

[5] Jung, H.-J.; Baker, Z.; Prasanna, V.: Performance of FPGA implementation of bit-split architecture for intrusion detection systems. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, duben 2006.

[6] Kroah-Hartman, G.: *Linux kernel in a nutshell*. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., 2007, ISBN 978-0-596-10079-7.

[7] van Lunteren, J.: High-Performance Pattern-Matching for Intrusion Detection. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, duben 2006, ISSN 0743-166X, s. 1–13.

[8] Michela. Becchi, P. C.: A hybrid finite automaton for practical deep packet inspection. In *Proceedings of the 2007 ACM CoNEXT conference*, CoNEXT '07, New York, NY, USA: ACM, 2007, ISBN 978-1-59593-770-4, s. 1:1–1:12.

[9] Sailesh Kumar, P. C.: Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In *In Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'06)*, 2006, s. 339–350.

[10] Seal, D.: *ARM Architecture Reference Manual*. Harlow, Essex CM20 2JE: Edinburgh Gate, 2001, ISBN 0-201-73719-1.

[11] Song, H.; Turner, J.; Lockwood, J.: Shape shifting tries for faster IP route lookup. In *Network Protocols, 2005. ICNP 2005. 13th IEEE International Conference on*, 2005, s. 367–377.

[12] Yang, Y.-H.; Prasanna, V.: Memory-Efficient Pipelined Architecture for Large-Scale String Matching. In *Field Programmable Custom Computing Machines, 2009. FCCM '09. 17th IEEE Symposium on*, april 2009, s. 104–111.

Příloha A

Obsah CD

- Zdrojový kód této práce v systému LaTeX.
- Vyhledávání vzorů pro OpenWRT - balík *pmatch* (semestrální projekt)
- Ukázkový soubor Makefile pro začlenění balíku do OpenWRT (semestrální projekt)
- Vyhledávání vzorů pro OpenWRT - balík *patternmatch*

Příloha B

Tabulky naměřených hodnot

MTU	PC – PC	DIR-825			Dockstar	
		switch	router	router+ fw	TX	RX
64	315.40	305.40	7.14	5.82	36.52	15.56
256	732.00	732.00	37.08	30.28	174.78	66.46
512	850.00	849.80	74.16	59.96	311.78	135.18
1024	920.60	920.40	148.40	117.40	609.22	343.66
1500	944.00	944.00	203.20	163.00	712.18	431.52

Tabulka B.1: Průměrné rychlosti v Mbps pro zařízení s 1 Gb Ethernetem, protokol TCP.

MTU	PC – PC	WAG 160N			Avila GW2348
		switch	router	router+ fw	
64	37.1	36.6	2.22	2.06	3.91
256	73.1	73.1	9.95	9.45	16.22
512	84.9	84.9	20.24	19.1	31.82
1024	92.1	92.1	40.56	38.42	58.36
1500	94.5	94.5	57.8	54.78	80.2

Tabulka B.2: Průměrné rychlosti pro zařízení s 100 Mb Ethernetem, protokol TCP

MTU	PC – PC	DIR-825			Dockstar	
		switch	router	router+ fw	TX	RX
64	142.18	133.82	14.18	13.92	18.0	3.07
256	294.8	518.8	34.98	44.08	71.5	12.34
512	542	584.2	95.08	101.2	142.2	24.44
1024	640.8	682	195.4	192.6	246.6	78.72
1500	713.4	568.14	115.46	120.6	350.6	186.00

Tabulka B.3: Průměrné rychlosti pro zařízení s 1 Gb Ethernetem, protokol UDP

MTU	PC – PC	WAG 160N			Avila GW2348
		switch	router	router+ fw	
64	48.0	48	3.45	3.58	6.73
256	79.2	79.34	17.38	16.7	26.52
512	88.6	88.44	34.44	33.14	50.54
1024	93.9	93.8	67.64	65.54	78.92
1500	95.7	90.4	30.58	30.22	62.18

Tabulka B.4: Průměrné rychlosti pro zařízení s 100 Mb Ethernetem, protokol UDP

Test	*Dockstar*	*Dockstar - opt*	*Avila*	*Avila - opt*	*Linksys*	*Dlink*	*RouterStation*	*Esprimo*
BF Insert 8k	816	222	1694	447	2864	1206	1122	183
BF Insert 96k	12596	3518	26859	6844	45932	18542	17854	2692
BF Query 8k	529	151	1136	309	2001	773	744	118
BF Query 96k	7984	2261	17175	4659	29044	11710	11336	1675
CBF Insert 8k	803	227	2023	466	2983	1179	1138	187
CBF Insert 96k	12858	5639	27501	7196	47171	18811	18149	3001
CBF Query 8k	549	158	1174	416	2088	791	766	127
CBF Query 96k	8022	2318	17388	4837	29495	11707	11297	1844
CBF Remove 8k	404	113	869	224	1451	591	568	95
CBF Remove 96k	6399	1762	13767	3493	23480	9366	9086	1507
(2000 a 24000 položek)								
SST7 BGP 1009	120672	37629	47241	18142	72598	27665	29910	21070
SST7 BGP 10084	118797	41368	56733	22493	79946	32677	35652	21309
SST15 BGP 1009	125595	40682	52271	20319	76883	30901	33973	21557
SST15 BGP 10084	127370	43385	47127	18547	72616	28231	30568	23441
TBM4 BGP 1009	52759	16172	17041	5572	27285	10647	10418	8289
TBM4 BGP 10084	56849	20041	17114	5609	27340	10695	10452	8513
TBM5 BGP 1009	60398	18061	17086	5948	28084	11050	10764	10275
TBM5 BGP 10084	62246	19860	17211	5942	28164	11087	10824	10484
(25 000 položek)								
Delay DFA HTTP	74726	59753	763516	248119				24674
Delay DFA Jabber	758817	616739	13062213	2851986				234207
Delay DFA FTP	156859	149956	1984784	530061				54650
Hybrid DFA HTTP	76181	57438	843950	249121				24733
Hybrid DFA Jabber	732533	472886	12179236	2145252				174509
Hybrid DFA FTP	6023480	4204351	48231363	20756277				1621484

Tabulka B.5: Kompletní hodnoty testů algoritmů. Doby běhů programu jsou v us.